KAROLIS RYSELIS

ALGORITHMS FOR HUMAN BODY SEGMENTATION AND SKELETON FUSION

DOCTORAL DISSERTATION

Kaunas
2023

KAUNAS UNIVERSITY OF TECHNOLOGY

KAROLIS RYSELIS

ALGORITHMS FOR HUMAN BODY SEGMENTATION AND SKELETON FUSION

Doctoral dissertation
Natural Sciences, Informatics (N 009)

Kaunas, 2023

This doctoral dissertation was prepared at Kaunas University of Technology, Faculty of Informatics, Department of Software Engineering during the period of 2017–2022.

The doctoral right has been granted to Kaunas University of Technology together with Vytautas Magnus University and Vilnius Gediminas Technical University.

Scientific supervisor:
Prof. Dr. Tomas BLAŽAUSKAS (Kaunas University of Technology, Technological Sciences, Informatics Engineering, T 007).

Edited by: English language editor Armandas Rumšas (Publishing House *Technologija*), Lithuanian language editor Aurelija Gražina Rukšaitė (Publishing House *Technologija*).

The official defense of the dissertation will be held at 9 a.m. on 26 June, 2023 at the public meeting of Dissertation Defense Board of Informatics Science Field in M7 Hall at The Campus Library of Kaunas University of Technology.

Address: Studentų 48-M7, Kaunas, LT-51367, Lithuania.
Phone: +370 608 28 527; e-mail doktorantura@ktu.lt

Doctoral dissertation was sent out on 26 May, 2023.
The doctoral dissertation is available on the internet http://ktu.edu, at the library of Kaunas University of Technology (Donelaičio 20, Kaunas, LT-44239, Lithuania), at the library of Vytautas Magnus University (Donelaičio 52, Kaunas, LT-44244, Lithuania), and the library of Vilnius Gediminas Technical University (Saulėtekio 14, Vilnius, LT-10223, Lithuania).

KAUNO TECHNOLOGIJOS UNIVERSITETAS

KAROLIS RYSELIS

ŽMOGAUS KŪNO SEGMENTAVIMO IR SKELETŲ SĄLAJOS ALGORITMAI

Daktaro disertacija
Gamtos mokslai, informatika (N 009)

Kaunas, 2023

Disertacija rengta 2017–2022 metais Kauno technologijos universiteto Informatikos fakultete, Programų inžinerijos katedroje.

Doktorantūros teisė Kauno technologijos universitetui suteikta kartu su Vytauto Didžiojo universitetu ir Vilniaus Gedimino technikos universitetu.

Mokslinis vadovas:
prof. dr. Tomas BLAŽAUSKAS (Kauno technologijos universitetas, technologijos mokslai, informatikos inžinerija, T 007).

Redagavo: anglų kalbos redaktorius Armandas Rumšas (leidykla „Technologija“), lietuvių kalbos redaktorė Aurelija Gražina Rukšaitė (leidykla „Technologija“).

Disertacija bus ginama viešame informatikos mokslo krypties disertacijos gynimo tarybos posėdyje 2023 m. birželio 26 d. 9 val. Kauno technologijos universiteto Studentų miestelio bibliotekoje, salėje M7.

Adresas: Studentų g. 48-M7, Kaunas, LT-51367, Lietuva
Tel. +370 608 28 527; el. paštas doktorantura@ktu.lt

Disertacija išsiųsta 2023 m. gegužės 26 d.
Su disertacija galima susipažinti interneto svetainėje http://ktu.edu, Kauno technologijos universiteto (K. Donelaičio g. 20, Kaunas, LT-44239, Lietuva), Vytauto Didžiojo universiteto (K. Donelaičio g. 52, Kaunas, LT-44244, Lietuva) ir Vilniaus Gedimino technikos universiteto (Saulėtekio al. 14, Vilnius, LT-10223, Lietuva) bibliotekose.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Relevance of the work

Various devices are used to capture diverse types of image-like information – RGB, depth, infrared data, and others. RGB and depth data are often used for semantic segmentation. With the privacy issues being raised more and more commonly, RGB cameras capture a lot of data and pose security threats [1]. Depth data, on the other side, carry less sensitive information, especially when humans are monitored. Depth data segmentation is usually one of the building blocks in real-world applications. This is a tempting area of research because low-cost devices such as depth cameras [2] or lidars [3] are widely available. Segmentation of spatial data [4] provided by depth sensing devices is researched a lot. It is also applied in multiple areas of research such as 3D face recognition [5], fall detection [6], evaluation of upper extremity characteristics [7], fitness applications [8, 9], exercise coaching [10], industrial worker activity monitoring [11], robotic applications [12], obstacle detection for the visually impaired [13], anthropometric measurements [14], posture recognition [15], general body tracking [16], and even image encryption [17]. However, there are multiple problems with depth data processing depending on the area of application.

The fundamental issue of object segmentation itself has gained extensive attention, especially with machine learning solutions [18]. One of the key components in this process is assigning labels to pixels, which usually means the type of object that a particular pixel belongs to [19]. Deep learning methods, especially convolutional neural networks (CNNs), are widely applied in object segmentation. They have largely contributed to the amount of research dedicated to this field. One of the issues is understanding the images at the semantic level, however, only recent solutions are practical to solve this issue [20]. Single-class (binary) segmentation is also sometimes useful. It is applied in areas as different as cloud segmentation [21], medical images [22], or human body segmentation [23].

If it is known that a certain object is already in the scene, the only issue to solve is segmenting the object from the background. This could be done by the already existing multi-class semantic segmentation systems, however, they are more complex than needed, which means that they are also more difficult to train. The issues are acknowledged by Shazeer et al. [24] who suggest a solution to disable some parts of the network, or Huang et al. [25] who suggest a solution to train very large networks more efficiently. However, the training time for large neural networks remains an issue, so smaller architectures are still required to reduce the time to deploy the CNN-based solution for some domains. Likewise, application domains are diverse [26]. However, the current state-of-the-art networks are very complex. *VGG-16* is a network for image classification. It is the base for many modern image segmentation networks. This network has 133M trainable weights [27]. It means that a huge amount of images is required to train such a network. The original research uses 1.45M images for training, testing and validation. Since this is a supervised learning technique, all images must have labels that are provided at least partially by a human. Segmentation tasks require much more human input for data labeling because, instead of assigning classes, part of the image has to be marked in some way. Large neural networks for segmentation also tend to have high inference times. State of the art shows that modern most powerful GPUs can only process relatively small images in real time (up to 256×128 resolution), while classic neural networks take up to 180 ms per image [28]. Larger images are more resource-demanding as the *SegNet* neural network processes a 1920×1080 image in 637 ms when using *NVIDIA Titan X GPU*. However, smaller architectures have been shown as viable [29]. They make it possible to process larger images faster. This enables new applications of such architectures – they can process data in real time, reduce hardware requirements, improve power efficiency, and require less data to train them.

Datasets suitable for image segmentation are therefore even more difficult to prepare. Noh et al. used a dataset of 12.3k images [30] to train their *Deconvnet* architecture. The dataset is called *PASCAL* and was already 5 years old at the time of the research [31]. It was collected by selecting specific images available online and then manually annotated by a team of annotators. Since the dataset was not large in the first place, Deconvnet training was also a complex procedure. It involved batch normalization to help the network escape the local minima and the creation of another sub-dataset from the PASCAL dataset by limiting variations in space to help the network capture the relevant properties of objects quicker, and only then learn spatial transformations. If datasets for segmentation had been easier to obtain in the first place, the PASCAL team could have created a larger dataset which, in turn, would have either made training Deconvnet easier, or applied techniques would have been more effective due to a larger set of training images. As another example, Xu et al. required over 43k annotated scans to conduct their research [32]. *ShapeNet* is a dataset which consists of 3M models, however, at the time of release, only 220k models were annotated [33].

The datasets required for segmentation from video sequences are even more demanding. For example, a usual frame rate of modern cameras is 30 frames per second. A one-minute-long video consists of 1800 frames. If such data has to be prepared to train a neural network, it takes an infeasible amount of manual work to mark foreground-background masks or object-label masks. However, such datasets are required for such applications as like real-time person segmentation in a browser [34].

The segmented human body can be used as a building block in a larger processing pipeline. Segmented point clouds were used to approximate human pose likelihoods [35]. Depth images were combined into a 3D human body model, which also includes segmenting both images to extract the human body [36].

In addition to RGB and depth data, the *Kinect* sensor also provides its skeletal information output with 25 joints; however, this output is not accurate in many cases [37]. This skeletal stream is used in numerous applications. They include human tracking solutions for activity recognition [38, 39], medical applications like balance recovery assistance [40, 41], postural control [42, 43], and assistance with Parkinson's disease

[44]. Other applications include gesture recognition [45]. *Kinect* is also used in multi-sensor tracking systems with wearable sensors to increase the tracking accuracy [46]. All the presently mentioned areas of application rely on the high quality of skeletal tracking. This problem may be solved by utilizing multiple sensors, however, two new problems arise. First, each sensor has its coordinate system, and, therefore, the sensors must be calibrated [47]. Next, the skeletons have to be fused into one [48]. However, none of these problems has a single best solution as the state of the art provides wildly different solutions.

1.2. Problem statement

Binary depth image segmentation is a problem of obtaining a mask for a depth image representing the 'foreground object' and the 'background'. When applied to the human body, it constructs a binary mask which indicates the pixels that belong to a human body. This is relevant in scenarios where the human body should be extracted from the image – it can then be used instead of a green screen, for tracking the human body in games, for such medical applications as physical rehabilitation, or sports activities like yoga. The applications, however, are not in the scope of this dissertation.

Binary depth image segmentation can be formulated as a problem where a binary mask B has to be acquired for the depth image D:

$$D = S(B) \tag{1.1}$$

Here, S is the segmentation function or algorithm, D is a depth image and B is a binary image with two values – the object pixel or the background pixel. This function can be defined in various ways and have more parameters depending on the area of application. This research focuses on the following sub-problems and provides a framework for solving them:

- Computer-aided semi-automatic segmentation;
- Fully-automatic segmentation.

Both types of segmentation may focus on general segmentation which can work with any type of object or focus on segmenting a specific type of object. General segmentation does not know in advance which object it should pick up; therefore, it needs extra parameters to specify some information that could define the desired object. Fully-automatic segmentation, by definition, cannot have extra parameters, it has to infer everything from the data. In both cases, this research focuses on binary segmentation where the object is a human body. This work focuses on applying semi-automatic segmentation to prepare datasets for supervised segmentation neural networks, and fully-automatic segmentation is meant to be applied in a larger depth data processing pipeline as a building block. Computer-aided semi-automatic segmentation research focuses on the performance of the algorithms to accommodate the final goal – reduction of the total time it takes to prepare a binary segmentation dataset. Fully-automatic segmentation focuses on smaller architecture sizes to allow shorter training times, a lower amount of training data, and, potentially, lower inference times with larger images.

More formally, computer-aided semi-automatic segmentation can be defined as a function with parameters of extra information and hyperparameters of a specific algorithm compared to the function S from Formula (1.1):

$$D = S_c(B, m, h) \tag{1.2}$$

Parameter m is provided from outside, for example, manually by a human, while h is a set of algorithm-specific hyperparameters. Fully-automatic segmentation, by definition, does not have the m parameter:

$$D = S_a(B, h) \tag{1.3}$$

Skeleton fusion is another problem related to human body tracking. It is used for similar applications, however, the tracking principle is different – instead of working with binary foreground-background masks, it operates with pre-acquired skeletal data from human tracking devices and aims to improve the accuracy compared to single-device tracking.

More formally, skeleton fusion is a problem where, given a human body in a 3D space monitored by multiple sensors from different angles, a single super-skeleton has to be constructed, desirably with better accuracy than when using a single camera. Skeleton fusion consists of two sub-problems: skeleton transformation to a common coordinate system, and skeleton merge that outputs a single skeleton. Given a set of skeleton representations from different angles $K = \{K_1, K_2, \ldots, K_n\}$, a transformation function T and a merge function M should be proposed such that a super-skeleton K_s is found with improved accuracy:

$$K_s = M(T(K)) \tag{1.4}$$

The goal of this work is:

- to provide an algorithm that outputs the result of the function S_c that could reduce the total time taken to prepare datasets for binary segmentation training compared to fully-manual segmentation;
- to provide a lightweight neural network architecture as a function S_a with a lower number of parameters and a lower inference time than neural networks for multi-class RGB segmentation;
- to provide functions T and M that would reduce fluctuations with the triple *Kinect* setup compared to a single *Kinect* device.

1.3. Tasks

The following tasks have to be accomplished to solve the problem of this dissertation:

1. Analyze the already existing solutions for geometrical image segmentation, machine-learning-based image segmentation, and skeleton fusion algorithms;
2. Propose performance-improving modifications for the geometrical image segmentation algorithm selected based on the analysis of the already existing solu-

tions;
3. Propose a machine learning model for automatic image segmentation;
4. Propose a novel skeleton fusion algorithm;
5. Evaluate all created algorithms in terms of accuracy and performance.

1.4. Scientific novelty

This dissertation provides the following scientific novelty:

- A bounding-box-based segmentation algorithm that is faster than Euclidean-clustering-based [49] by 367 times with the selected dataset.
- A recursive 2-Means split algorithm with a random forest classifier for split acceptance that reduces under-segmentation by 2.5 times on average on a selected dataset.
- A fast multiple *Kinect* skeleton calibration and fusion algorithm which reduces skeletal data fluctuations produced by *Kinect* from 10% to 2%.
- A novel abridged convolutional neural network architecture for human body segmentation which requires 27 times fewer parameters and has a 2 times lower inference time than the *SegNet* neural network [50] with a 40% larger image size.

1.5. Thesis statements

- The total human-supervised point cloud segmentation time can be reduced by 66 times by applying an optimized bounding-box-based technique compared to full-manual segmentation and by 12% compared to the use of Euclidean-clustering-based segmentation [49].
- An abridged binary depth segmentation neural network can achieve similar accuracy to the state-of-the-art large neural networks for multi-object RGB segmentation.
- Bounding-box-based under-segmentation area can be reduced by 2.5 times by cutting part of the point cloud while using the proposed recursive 2-Means split and the proposed metrics of the split.

1.6. Practical value

The proposed semi-automatic segmentation algorithms with their improvments have been implemented in a software tool to mark binary masks for *Kinect* depth images. They helped to reduce the total computer-aided segmentation time and made it feasible to create the binary masks for a dataset of 220k depth images by only two people. Therefore, the primary application of the algorithms is to be implemented in data labeling for supervised machine learning architectures. However, they could also potentially be applied for unsupervised clustering tasks. This tool carries the most scientific novelty as there are novel variations of algorithms in use.

The proposed *Agrast-6* neural network architecture could be applied in larger depth data processing pipelines where the network could be used as one of the building blocks. One potential application of such a neural network is human silhouette comparison by using such metrics as the Hausdorff distance. This could be applied in yoga training, rehabilitation activities, and other human tracking. Trained *Agrast-6* model is the ultimate result of this dissertation as all research about segmentation leads to this model and its training.

Lastly, the proposed skeleton fusion algorithm has been applied for human motion analysis [9]. More accurate measurements were made to determine the load of human joints during various activities. The algorithm could also be used as a means of capturing a more accurate representation of a human skeleton by using multiple *Kinect* devices.

2. STATE OF THE ART

This chapter presents the current state of the art of the areas related to this dissertation. The whole research focuses on depth information processing.

Section 2.1 introduces the state-of-the-art research related to depth sensors and depth data processing methods. First, depth-sensing devices are presented and compared. These devices capture the depth and human body skeletal information. Both skeletal (transformation and fusion) and depth (depth maps, 3D search trees) data processing is reviewed in this section. Finally, noise reduction methods for these devices are presented. There are several metrics to assess the accuracy of segmentation. They are presented in Section 2.2. The main focal point of this dissertation is human body segmentation. While segmentation and clustering are different tasks, they are both related. Therefore, segmentation and clustering techniques are overviewed in Section 2.3. This section reviews geometrical segmentation techniques, which are used as a baseline for semi-automatic segmentation research, as well as unsupervised clustering, full manual segmentation, and neural-network-based segmentation. Random forest techniques may also be used in combination with other techniques to solve classification tasks. They are overviewed in Section 2.4. Finally, Section 2.5 concludes this chapter and provides an overview in terms of whichever existing solutions can be used, adapted, or used as a baseline for the proposed segmentation framework or its components.

2.1. Depth data acquisition and processing

2.1.1. Depth sensing devices

There are two types of data analyzed in this work – depth data (distances between a sensor and an object at all pixels) and skeletal data (a set of human joint positions in a 3D space monitored by a sensor). The only widely-available device providing skeletal data is *Microsoft Kinect*, but there are many available depth-sensing devices. This section overviews the most-commonly-used devices for capturing depth data – different versions of *Microsoft Kinect* and *RealSense* devices. These devices have different resolutions, different amounts and even types of noise. Choosing a better device to capture the data might make it easier to work with the data. Properties of *Kinect's* skeletal stream are also reviewed. Stereo cameras are also used in computer vision, and, therefore, they are discussed in this section as well.

2.1.1.1. *Microsoft Kinect* and depth data

The most popular depth-sensing device is *Microsoft Kinect*. It was initially released in 2010 and made available for PC (Windows) in 2012. Its depth camera had a depth resolution of 320×240. It was the first depth device to become widely available among home users. *Kinect* was included in the Guinness Book of Records as the "fastest selling gaming peripheral" with 8M units sold in the first 60 days [51]. It was superseded by its next iteration *Kinect 2* in 2014. This variant offered a better depth resolution of 512×424 and has been proven to be more accurate by different researchers. It yielded better accuracy for karate technique recognition [52], respiratory motion tracking [53], and rehabilitation of stroke patients [54]. Both versions of *Kinect* have been investigated and used by a plethora of researchers and across extensive areas. *Azure Kinect* is the latest iteration of the sensor released in 2019, however, it does not seem to be utilized as much as *Kinect 2*. A disadvantage of *Azure Kinect* is its non-rectangular fields of view (circular or hexagonal) [55]. While hexagonal neural networks have been proposed [56, 57], they are not widely supported by standard machine learning frameworks. An alternative is to utilize a larger rectangular neural network, however, this approach is less efficient because part of a depth image will always be empty. It was also shown not to have better human tracking accuracy for mid and upper body parts [58].

The *Kinect 2* sensor provides multiple data sources – RGB, depth, IR video streams. Its SDK also provides a human body index stream that tracks up to 6 people at once. Skeletal tracking with 25 supported points is also available. However, the reliability of the latter streams is questionable as the device is meant primarily for games where the human is in an upward pose. This is also apparent in related studies using this type of data. The *Kinect 2* sensor was used for a virtual dressing room, and measurement errors were within 10%, however, the tracked subjects were all in standing poses [59]. It was also implemented in a sign language recognition system utilizing skeletal tracking, however, 15% top-1 match accuracy is not high [60].

All versions of the sensor have some type of noise in their depth images. Mallick et al. analyzed the types of noise present in *Kinect v1* [61] and found that 9 types of noise should be accounted for. Depth image quality was shown to be improved by a hole-filling pixel interpolation and a bilateral filter [62, 63]. The tracking quality was also shown to be affected by the background clutter and lighting conditions for the older *Kinect* sensor [64]. However, the noise levels of *Kinect 2* and *Azure Kinect* are much lower and more predictable as shown by Tölgyessy et al. [55]. *Kinect 2* was shown to have varying levels of lateral and axial noise [65] as well as spatial, and temporal artifacts and holes [66] and scene-dependent pixel-wise dropouts and distortions [67]. It also adds shadows on the object's edge, which affects the shape recognition [68]. Another general issue with depth-sensing devices is the capturing of quick topological changes [69]. This is an important drawback for this research because it limits the application areas to static scenes.

2.1.1.2. *Microsoft Kinect* and skeletal data

Kinect sensors also offer their solution for multi-human segmentation and the estimation of specific body points. Unfortunately, this seems to work well only under the right conditions. Object occlusion is one of the problematic areas for the *Kinect* software. This problem may be tackled by applying reliability measurements for tracked body parts and correcting the pose according to natural human body constraints [70]. Another solution is suggested by Ho et al. [71]. The researchers also suggest a reliability assessment methodology for detecting human body joints. However, this proposition relies on the related depth and RGB data rather than predefined human body model constraints. Occlusion of the human body may also be solved by utilizing neural networks. A deep recurrent hierarchical neural network was proposed to eliminate posture detection problems [72]. It achieved over 91% accuracy, however, it is a resource-demanding solution capable of processing only 10 frames per second for sitting poses. Another problem is the low accuracy or even completely invalid results when the tracked human is in a non-standard pose such as standing on his hands or bent forward [37, 73]. This shows that standard algorithms have a limited area of application and either must be used together with some correcting techniques or replaced completely if the original performance is not satisfactory. This was proven possible by *PointSkelCNN* – this is a deep learning architecture to extract the human body skeleton points [74]. Skeleton extraction was also proposed via an adversarial architecture [75].

2.1.1.3. ***Microsoft Kinect's*** data capturing principles

Kinect devices capture depth data by using infrared illumination. The device illuminates the scene, and an infrared sensor captures the reflected infrared light. The illumination forms a pattern that is deformed by the illuminated structures in the scene. These deformations are then interpreted by measuring the light phase shift [76]. The phase shift is caused by the time it takes for the light to travel to the object and back. The device can then compute the distance that the light traveled. This approach is called the time-of-flight approach [77]. The device has the means to distinguish its reflected signal from an external infrared source. This is accomplished by using several correlation measurements with varying illumination. Device interference is another possible source of errors, however, this is solved by using different modulation frequencies for different devices. Despite that, some decrease in the image quality is observed when multiple sensors are used [78]. This decrease does not limit the usage of multiple *Kinect* devices in the same environment as depth loss is minimal. Depth values at object boundaries are usually distorted because the light reflects from different distances in the area of the same pixel. These pixels are referred to as 'flying pixels'. This is also related to light scattering, where the light can be reflected more than once before reaching the sensor. This leads to incorrect depth values and also produces flying pixels. *Kinect* cameras are also affected by motion because the time it takes for the light to arrive at the sensor depends on the distance to the object, which may change during that time. The device tries to solve this by using several correlation images per frame. The sensor also provides no depth data if an object absorbs infrared light as no light travels back to the sensor [78, 79]. In the light of this research, a case of black jeans is important to note [80] since this material absorbs infrared illumination.

2.1.1.4. Intel ***RealSense***

Intel released a series of depth-sensing devices called *RealSense*. The camera models available in 2022 are *D455*, *D435*, *D415* and *D405*. They all work in both indoor and outdoor settings but have different working ranges – *RealSense D405* is adapted to short-range videos (up to 50 cm) while *RealSense D455* has the longest range of 6 m. Earlier devices include the *L500* series and *SR300*. Even earlier devices that had their support dropped with *RealSense SDK 2.0* in 2018 are *F200*, *R200*, *LR200* and *ZR300*.

RealSense and *Kinect* sensors were both used in the same research for sign language recognition and provided similar results with *RealSense* having a slight edge [81]. Unfortunately, the researchers did not state which version of *RealSense* they used. Neupane et al. conducted an in-depth sensor comparison and found that *Azure Kinect* was the most accurate, *Kinect 2* was less accurate, and different variants of *RealSense* were less accurate than *Kinect 2* [82]. Yin et al. showed that *RealSense R200* device suffers very heavily from noise in human body tracking and introduced a whole pipeline to reduce it [83]. *RealSense D435* was also shown to have higher noise levels than *Kinect 2* [84, 85].

2.1.1.5. Stereo cameras

Another type of device is the stereo camera. This is a system of two cameras that can both view the same volume of space similar to human eyes. They are widely used in robotics. The applications in this area include SLAM (*Simultaneous Localization and Mapping*) [86] and robot position estimation [87]. One of the open-source solutions for SLAM, *ORB-SLAM2*, also integrates algorithms for SLAM for stereo cameras [88]. Depth stereo cameras are also used for robot position estimation [89]. Specialized devices like thermal stereo cameras are also used [90]. One of the main challenges with stereo cameras is camera calibration since two devices are involved. This is a process where the camera parameters are determined, including the positions of the cameras. Specialized patterns are often used to solve this problem, the most common being a calibration grid. They are analyzed by using different geometrical methods [91, 92, 93], and other computer vision methods are also involved, for example, Otsu thresholding [94]. However, these devices are more prone to image blurring while moving [95]. Since they are widely used in SLAM, this has to be solved by image deblurring techniques. While stereo cameras are useful in SLAM, they are not widely used in other computer-vision-related research. There is some research on image and video segmentation using depth cameras [96, 97], however, the proposed methods are usually geometrical. Machine learning methods, such as neural networks are hard to adapt since there are no machine learning architectures for camera calibration. End-to-end learning solutions are also, to the best of the knowledge of the author, not yet available, because the nature of ever-varying extrinsic camera parameters adds a whole extra dimension of features to learn. Therefore, stereo cameras seem to only be easily applied in robotics, mainly SLAM.

2.1.1.6. Summary

To sum up, *Kinect* devices provide the best quality of depth images, however, the latest *Azure Kinect* has a hexagonal image shape, which makes it more difficult to process it compared to rectangular images. Images from stereo cameras need calibration, which complicates the processing of their images. *Kinect 2*, therefore, is the best fit for capturing depth data for the purposes of this research. *Kinect's* working principles suggest that it may not be accurate in all conditions, especially when observing light-absorbing materials. *Kinect's* skeletal stream is inaccurate, however, its accuracy may be improved by combining the output of multiple sensors.

2.1.2. Combining skeletal data from multiple sensors

This research aims to improve the shortcomings of the *Kinect* device's skeletal data accuracy by using several sensors and combining the outputs. This involves two problems – transforming the outputs to a common coordinate space, and then fusing the transformed outputs into a single, presumably, more accurate skeleton representation.

2.1.2.1. Skeleton transformation

The most common methods to transform skeletons into a common coordinate space are:

- Iterative closest point algorithm;
- Matrix transformations;
- Marker-object-based transformations;
- Geometric transformations;

The main idea of the iterative closest point algorithm [98] is minimizing the difference between two point clouds P_1 and P_2. It consists of the following steps:

1. Find the closest point in P_2 for each point from P_1.
2. Estimate rotation and translation by using a distance minimization technique.
3. Transform P_1 according to the obtained translation and rotation.
4. Repeat from Step 1.

This technique was employed in dance analysis using multiple *Kinect* devices [99], 3D object detection in a kitchen environment [100], and 3D face recognition [101]. However, this algorithm involves multiple iterations, and there is no best indicator of convergence.

A matrix-transformation-based algorithm is proposed in the human motion estimation research [102]. The researchers observed that the torso is a rigid part of the human body and is usually detected with high confidence. Therefore, it is selected as a reference joint, a matrix of inter-skeleton differences is constructed, and a rotation matrix is computed by using singular value decomposition. The inter-skeleton difference matrix H is computed as

$$H = \Sigma_{l=1}^{3}(x_l^{k_0} - \overline{x^{k_0}})(x_l^k - \overline{x^k})^T \tag{2.1}$$

Here, k_0 is the reference skeleton, x_l^k is the coordinates of joint l of skeleton k ($l \in \{1, 2, 3\}$ and represents the torso and two shoulders). Singular value decomposition yields values $[U, S, V]$, and VU is used to construct a 4×4 rotation matrix

$$T = \begin{pmatrix} - & I_{3\times3} & \overline{x^{k_c}} \\ & 0 & 1 \end{pmatrix} \begin{pmatrix} - & VU^T & 0 \\ & 0 & 1 \end{pmatrix} \begin{pmatrix} - & I_{3\times3} & -\overline{x^{k_0}} \\ & 0 & 1 \end{pmatrix} \tag{2.2}$$

All skeletons are then rotated by using this matrix. Unfortunately, the authors do not disclose the meaning of k_c.

One more possibility is the use of marker objects in the scene. The viability of this approach was demonstrated in the *Livescan3D* system to acquire data from multiple *Kinect* sensors [103]. An easily recognizable object is placed in the scene and then found in the output of each sensor. Then it is used as a reference to find the correct transformation. Since this object is the key to data transformation, it has to be visible by all sensors and not occluded, which limits its area of application. The transformation may then be refined by using the iterative closest point technique. A similar approach was proposed in a 3D human skeleton tracking solution [104]. The researchers used a cuboid calibration box with different images on different sides of the box. An iterative algorithm for skeleton transformation is used as well.

Geometric transformation is also possible as shown by automatic multiple *Kinect* cameras settings for walking posture analysis [105]. A rotation matrix R_{xyz} for angles θ_x, θ_y, and θ_z is required as well as a translation matrix T for translation distances t_x, t_y and t_z. The full transformation matrix M can then be defined as

$$M = T \cdot R_{xyz}(\theta_x, \theta_y, \theta_z) \tag{2.3}$$

Transformations for x, y, and z coordinates are then defined as

$$\begin{aligned} x' = \ & x(\cos\theta_z \cos\theta_y) + y(\cos\theta_z \sin\theta_y \sin\theta_x - \sin\theta_z \cos\theta_x) + \\ & z(\cos\theta_z \sin\theta_y \sin\theta_x - \sin\theta_z \cos\theta_x) + t_x \end{aligned} \tag{2.4}$$

$$\begin{aligned} y' = \ & x(\sin\theta_z \cos\theta_y) + y(\sin\theta_z \sin\theta_y \sin\theta_x + \cos\theta_z \cos\theta_x) + \\ & z(\sin\theta_z \sin\theta_y \cos\theta_x - \cos\theta_z \sin\theta_x) + t_y \end{aligned} \tag{2.5}$$

$$z' = x(-\sin\theta_y) + y(\cos\theta_y \sin\theta_x) + z(\cos\theta_y \cos\theta_x) + t_z \tag{2.6}$$

A similar geometric solution is used for dance analysis, however, the authors do not reveal the fine details of the transformation [106].

The presently mentioned techniques can also be combined as shown by the fusion of information from multiple *Kinect* sensors research [107]. They use marker objects with the iterative closest point algorithm. After that, they refine the transformation parameters by using a technique similar to that defined in Equations (2.1) and (2.2).

The skeleton transformation techniques are summarized in Table 2.1. Iterative algorithms (iterative closest point and rotation-matrix-based) require multiple iterations until the result skeleton converges. The amount of iterations is unknown beforehand. On the other hand, matrix-transformation-based and marker-object-based algorithms have a fixed amount of computations to perform. The iterative closest point algorithm seems to be the most widely used, and it provides good accuracy compared to other analyzed methods, but it is slower due to its iterative nature. The matrix transformations method requires singular value decomposition which has a time complexity of $O(mn^2)$, where m is the larger dimension size of the matrix, and n is the lower size [108]. For *Kinect 2* skeletal stream, $m = 25, n = 3$. Geometric solutions do not suffer from the disadvantage of high computational complexity, however, they require calibrated sensors.

This review suggests that a rigid body part may be used to calibrate the sensors

Table 2.1. Comparison of transformation approaches

	Iterative closest point[98]	Matrix transformations[102]	Marker object based[103]	Rotation matrix based[105]
Base object	Whole skeleton	Rigid body part	Known extra object	Whole skeleton
Amount of calculations	Higher	Higher	Lower	Higher
Reported accuracy	94%[99], improved indirect output accuracy from 62% to 69%[100]	89% (indirect metric)	Not provided	10 cm error
Disadvantage	Performance (iterative)	Performance (singular values)	Requires extra objects in scene	Performance (iterative)

similarly to the matrix-based proposals, but the transformation itself could be found via geometrical means to maximize the performance.

2.1.2.2. Skeleton fusion

The second step – skeleton fusion – determines the final position of the skeleton joint based on the transformed values from all *Kinect* devices. The simplest option is to compute the average joint position across all sensors [105]. Another option is to estimate the confidence levels. Dance analysis research proposes a solution of this type [99]. Fusion is performed by selecting the best skeleton by the confidence level, and the joints with non-high confidence are combined with the same joints from the other skeletons by using a weighted average based on the confidence levels. Unfortunately, the paper does not go into much detail on the confidence level estimation. Human motion estimation research proposes a similar approach to this problem, however, the researchers have a slightly different task – to eliminate occlusion by using several sensors [102]. Similarly to the previous paper, confidence levels are used to fuse the skeletons. An estimate of the overall activation of the joint is computed as the standard deviation from its position derivatives over time. Another indicator for confidence was the depth value at the location of the joint – in the case of occlusion, it will be different from the rest of the body. One more possible approach to determine the confidence is to estimate the measurement noise based on the predicted state and joint motion continuity [109]. The prediction is made by using the Kalman filter which minimizes estimate covariance. Motion continuity is computed from the joint positions over time for each sensor separately. Then all sensors vote whether the movement was fast or slow. A similar technique with Kalman filters was also implemented for human-robot collation collision detection [110] and gait analysis [111].

2.1.3. Depth-related data structures

There are multiple ways to represent depth data. The most common data structures are these:

- Depth map;
- Three-dimensional binary tree;
- Octree.

These data structures have different properties and might be useful for solving different problems. They are reviewed in this section.

2.1.3.1. Depth maps

Depth data can be represented in various formats. The most commonly used format returned by depth sensors is a depth map. It has an immediate advantage because it does not need to be constructed – this is a direct output from the sensor's software. This is a picture-like data representation which consists of one channel, and the values of pixels represent a distance from the sensor rather than color. Depth maps share many properties with the regular RGB and other images, and, therefore, are a viable input for convolutional neural networks. Couprie et al. applied neural networks to perform the semantic segmentation of depth images [112]. Their machine learning model achieved 52% pixel-wise accuracy when classifying depth scenes into 14 categories. Xu et al. have shown that the depth data from LIDAR sensors can be segmented by using their proposed neural network architecture *SqueezeSegV3* [32]. This network consists of 5 convolutional stages and an upsampling layer. This solution yields 55.9% mean intersection over union (mIoU) accuracy. Another CNN-based solution is RGCNN [113]. This model is also used for semantic segmentation and reaches 84.3% mIoU, ranging from 44 to 95% for different classes. Wang et al. suggest their RGBD segmentation CNN (convolutional neural network) which yields 49-61% accuracy [114]. This model works with any object. Depth images of pigs were successfully used with the *YOLO9000* neural network to find bounding boxes of them so that to identify touching pigs in a closed environment [115].

Depth maps can be processed by using non-machine learning methods similar to regular images as well. Mussi et al. investigated ear segmentation from depth map images [116]. The researchers used different classic algorithms for image processing – Canny detector, the local adaptive threshold as well as analyzing histograms to determine contrast corrections. They managed to achieve a similarity of 98% and more in ear segmentation. Their results show that the classic image processing algorithms apply to depth images as well. Li et al. proposed a three-step algorithm based on region growing, which is another algorithm widely used for RGB image segmentation [117]. Their area of application is the plant leaf segmentation, and it reached 95-96% F-measure. A combination of a watershed and K-Means clustering has been used to segment LIDAR point clouds [118]. This approach increased the segmentation accuracy of the available watershed-based solutions. Li et al. proposed a technique based on human body part models and graph cuts and computed pixel probabilities [119]. This works directly with depth data. These examples demonstrate that the classic RGB image analysis algorithms can be successfully applied to depth maps as well.

Depth maps are most widely used together with other image or image-like information. Common candidates are RGB images and thermal images. Using multiple types of data tends to increase both the accuracy and complexity of solutions. Hsieh et al. have applied RGBD data to apply semantic segmentation of the sidewalk [120]. They adapted a Fast-SCNN neural network that was originally intended to use high-resolution RGB images to work with RGBD images [121]. Their research showed 95% pixel-wise accuracy in semantic segmentation. Kang et al. also used RGBD data with their neural network add-on for semantic segmentation [122]. Their results also suggest that different types of neural networks can be used with both RGB and RGBD data. Sun et al. used RGBD and long-wave infrared data for hand segmentation [123]. This research showed a 5% accuracy increase in segmentation over using only RGB data. Palmero et al. also used RGBD and thermal data for human body segmentation [124]. They proposed 4 different descriptors and fused them by using random forest. Their approach showed an accuracy of up to 79%, which confirms that a combination of different techniques can be applied for segmentation. Huang et al. implemented a robust human body segmentation algorithm based on part appearance and spatial constraint [125]. They create body part models and compute the probability of the pixels belonging to each part, then construct a cut-graph, and segment the image based on it. The solution works with RGB images and is created specifically for human body segmentation. Zhao et al. combined it with RGB data to recognize human activity [126]. The combination of RGB and depth data is also used in a variety of other solutions for object detection [127], and especially semantic object segmentation [128, 129, 130, 131].

Despite being more prevalent, adding extra information compared to depth data involves some disadvantages. First, the data becomes more complex than depth-only. In the case of semi-automatic segmentation, one type of data is easy to process for a human (depth, IR, or RGB), however, their combination is difficult to interpret. Multiple modalities are introduced into manual segmentation workflows by displaying different data types as separate images [132], however, this increases the amount of manual work, and the segmentation outputs have to be merged afterwards, which is a new challenge. An alternative approach is to merge the modalities into a single representation [133]. However, this can greatly impact the data processing time depending on the merge algorithm. Ultimately, the quality of segmentation then also depends on the quality of the merger. The reduced amount of data tends to speed up automatic neural-network-based segmentation. However, a reduction in accuracy is not always observed. It was shown that changing RGB data to grayscale data did not reduce the object recognition accuracy [134]. Depth data, added to RGB images, tends to improve the data processing accuracy, however, the difference in accuracy is usually within 5 percentage points [135, 136]. Neural networks, on the other hand, have to either accept high-dimensional inputs or combine the results from different modality inputs, thus making the architecture larger, and therefore potentially slower and harder to train. The *Kinect* device captures depth data by illuminating the scene and collecting the reflected IR waves, while RGB and IR streams are captured by registering light and IR waves emitted by the object or reflected from the outside source. This makes the depth stream much less reliant on external factors and more consistent, thereby introducing less variance to the data. Meanwhile, low lighting conditions affect RGB accuracy [137, 138]. Therefore, depth data is analyzed in this research for segmentation purposes as the depth stream is more consistent regarding the external conditions, carries less information, which makes faster processing possible, and it is easier to analyze for the human than multi-modal data.

2.1.3.2. Three-dimensional binary search trees

Multidimensional search trees were first introduced by Bentley in 1975 [139]. This data structure has useful properties for point cloud analysis. It has $O(\log n)$ search complexity. This is important because two points that are in neighbor pixels in a depth map can be far away from each other in 3D space and vice versa. Space-local search is therefore slower in a depth map and faster in a three-dimensional tree. Therefore, three-dimensional trees are often used when the neighbor point search is important. Good search complexity is an important property because close points are the basis of a plethora of algorithms that have been developed to work with this data structure. Examples include segmentation [140, 113, 141], clustering [142, 143, 144], classification [145, 146, 147], and others.

A binary search tree is constructed by selecting a pivot and splitting the list of items into two parts – smaller than the pivot and larger than the pivot. The smaller elements are moved before the pivot, while the larger elements are moved after the pivot. This idea comes from the quick sort algorithm originally introduced by Hoare [148]. However, finding a good pivot influences the performance of both the tree creation and search. If a pivot is good, it splits the original list into almost equal-sized parts. This allows the creation of a nearly-balanced search tree that will, as a result, have nearly $O(\log n)$ search time. If a pivot is bad, the two parts of the list have very different sizes. In an extreme, where the minimum or maximum is selected, an unnecessary extra layer in the tree is added, which leads to increased search times. The best-case scenario is the median value because it splits the list into equal parts.

Unfortunately, finding a median in an unsorted collection is computationally expensive. In the most straightforward case, the list of items is sorted, and the middle element is selected. However, this usually leads to $O(n \log n)$ complexity for a single split. The splits are required for all layers of the tree except the last one (only one item, so nothing to split). Sorting the list is impossible for a multidimensional case since the ordering is different at different depths of the tree, so the list has to be re-sorted after each split. Therefore, there will be one $O(n \log n)$ sort for the first layer, two ($O(\frac{n}{2}) \log \frac{n}{2}$), and so on. The total complexity of all sorts will be $\Sigma_{i=1}^{\log n - 1} \frac{n}{i} \log \frac{n}{i}$.

One of the ways to solve this problem is to use a median-of-three algorithm [149]. It suggests selecting three random elements from the list and finding a median for them. This significantly reduces the amount of work to sort the elements, but the computational overhead is small compared to the actual median computation.

Other researchers have suggested different techniques to construct a k-dimensional search tree. Brown suggested pre-sorting data in all dimensions before building the tree [150]. The ordering is then preserved when constructing the tree. He has also shown that the complexity of this approach is $O(n \log n)$ as re-sorting of the whole tree is not required. It was also suggested to build a k-dimensional tree by utilizing GPU parallelism [151]. The researchers showed that it is possible to reach a speedup of a factor of 30.5 by using *NVidia GTX 660* GPU and their algorithm compared to a sequential algorithm on *Intel Core i7-960* CPU. Another GPU-based tree construction algorithm was able to construct the tree of 200k points in about 6 ms when using *NVidia Titan X* GPU [152]. The speedup is achieved by a massively parallel adaptive sorting algorithm. One more approach to the tree construction on GPU is to move all points to the leaves of the tree so that the internal nodes only contain split information [153]. A tree with 100k nodes is constructed in 321 ms when using this approach, however, the tree has 512 dimensions rather than 3. One GPU thread works for each split to create the initial representation, and the nodes are moved to the leaves after that. However, all approaches also require CUDA. In addition to that, if further processing has to be done by using the CPU, the tree has to be reconstructed back from the GPU representation. While this is faster than creating a full search entirely on the CPU, the reconstruction is still a task that has to be performed. ikd-tree was also suggested [154]. A partial search tree is constructed and then updated with new coming points. This process involves rebalancing the tree, and it also leaves data from the previous depth scans. Essentially, this approach updates an existing search tree. This avoids the problem of creating many search trees at the cost of extra used space and a slightly worse search performance.

2.1.3.3. Octrees

An octree is a data structure that divides the 3D space into cubical subsections [155]. It also has the same search complexity as the k-dimensional tree – $O(\log n)$. They are also widely used in similar areas, such as segmentation [156, 157, 158] or classification [159, 160, 161]. On the other hand, this data structure is not commonly used for clustering tasks. It is more difficult to build clusters in the case the points of the cluster fall into separate subdivisions of the octree.

An octree is constructed by dividing the whole space into 8 (usually) equal parts recursively. A challenge for octree construction is when to stop dividing the space. If more than one point stays in a ‘leaf’ octant, search complexity grows as it is not enough to only find an octant, but also to find a point in this octant. On the other hand, if there is at most one point per octant, it could lead to many empty octants in cases where 2-7 points remain in an octant, and it has to be subdivided into 8 sub-octants. This would lead to higher memory requirements per point on average. Indeed, high memory usage is a recognized downside of an octree [162, 163, 164].

2.1.3.4. Conclusion

It is evident from the state of the art that depth maps are a good fit for most of the problems. Their image-like properties make it easy to use them with neural networks. However, they are also different from images because they represent spatial data – it is not a flat image like RGB data. Search trees and octrees better represent the spatial aspect, however, search trees tend to use less memory while other properties are similar. Therefore, three-dimensional binary search trees could be useful in semi-automatic segmentation tasks, where spatial data analysis is required. Depth maps can be processed directly by neural networks, which makes them suitable for fully-automatic segmentation.

2.1.4. Noise reduction

Since regular image processing techniques can be applied to depth maps, noise reduction techniques also apply. Usually, one of the filtering algorithms is used because they are fast to compute and tend to provide decent results. Different noise reduction algorithms should be used for different types of noise and have different properties.

Mean and median filters are the simplest noise reduction techniques. They compute a kernel of a predefined size by averaging the values of computing a median of the values. These types of filters have been observed to work well with the salt and pepper type of noise [165, 166]. However, a small kernel size tends to not reduce the noise enough, while a larger kernel size loses more details of the original image. This problem has been addressed in different ways. Zhu et al. suggested computing the kernel size adaptively depending on the values and combining the median and mean filters [144]. This technique reduces the noise better while keeping more details of the original image. Another implementation of an improved median filter has shown good results even at high noise levels [167]. The algorithm may also be used in image segmentation to some extent. It was applied with Otsu segmentation to segment RGB images [168]. An image is segmented three times for each channel and then merged, however, the image remains distorted which is solved by applying a median filter. The median filter is useful for the salt-and-pepper type of noise reduction [169], however, this filter and its variations blur the edges of the image [170]. Both median and mean filters were also shown to successfully improve the peak signal-to-noise ratio (PSNR) for Gaussian and speckle noise, however, the median filter worked better in all experiments [171, 172]. This filter has $O(n^2)$ time complexity for an $n \times n$ sized image. It was also shown to fail at removing a large amount of noise [173]. However, variations of mean and median filters tend to work well only on some types of noise.

The Gaussian filter is a more complex algorithm of noise reduction. Instead of treating all neighbor pixels equally, this filter gives more weight to the closer pixels. The weights are defined by a Gaussian curve. This filter tends to be more effective than the mean or the median in most situations, however, it is more computationally expensive. The Gaussian filter was used to smooth intermediate data representation in plane detection in point clouds [174] and to pre-process images for the tree crown segmentation [175]. A Gaussian-based filter was shown to produce a better PSNR when reducing the speckle noise than the median filter [176]. It allowed more robust individual point contributions. This type of filter was also implemented in 3D point cloud matching [177]. Its purpose was smoothing voxels in a predefined voxel grid. However, none of those researches further investigated the effect of this filter on the accuracy of the experimental results.

Table 2.2. Comparison of common noise reduction techniques

Filter	Performance	Preserves edges	PSNR improvement
Mean	Very fast	To some extent	Lower
Median	Fast	To some extent	Lower
Gaussian	Slow	No	Higher
Bilateral	Very slow	Yes	Higher
Machine learning	N/A	Can learn specific types of noise	N/A

Another type of image noise reduction is a bilateral filter first introduced by Tomasi et al. [178]. It adds range weight to the Gaussian filter. This property enables the filter to better preserve the edges [179]. As a result, it is widely used in image segmentation. A variation of the bilateral filter was used as a building block in noisy image segmentation [180]. The filter was also implemented in a fuzzy C-Means segmentation algorithm [181]. A bilateral filter is suitable to point cloud denoising when high-frequency noise is present in the data [182]. It was also shown that the application of a bilateral filter reduced the error of the fetal length estimation using watershed segmentation [183]. Unfortunately, this filter is slower to compute than the Gaussian, median of mean filters. There were numerous attempts to make it faster by subsampling the image [184], decomposing the filter into a set of 3D box filters [185], or utilizing field-programmable gate arrays [186], however, they either sacrifice some accuracy, need other techniques in combination, or use custom-made hardware. For images sized 512×512, even the optimized approximated variant of this filter produces its output in 75-186 ms depending on the parameters [187].

Noise reduction is also possible by using machine-learning-based solutions. Adversarial neural networks are applied for noise reduction tasks. There are different approaches to solving this problem, however, noise reduction is usually integrated into the overall solution. One such example is an adversarial auto-refiner network that encodes the point cloud, and extracts and cleans up the features that are then processed further [188]. However, the noise reduction step using machine learning is a viable option when the data is processed further by using machine learning methods. Geometrical methods tend to work well enough and do not require a complex learning process. The comparison of all the reviewed noise reduction algorithms is presented in Table 2.2.

2.2. Point cloud similarity metrics

One of the most popular set similarity metrics is Dice's coefficient, also known as the Sørensen-Dice index [189]. Given two sets A and B, it is defined as

$$DSC = \frac{2|A \cap B|}{|A| + |B|} \tag{2.7}$$

It compares the sizes of the sets to the intersection of the sets. Therefore, if A and B are non-intersecting sets, their Dice's coefficient is 0, and if they are equal, Dice's coefficient is 1. However, this metric gives a low penalty if the sets have different sizes. If A is a subset of B representing a fraction of k points, the coefficient would be equal to

$$DSC = \frac{2k|B|}{(1+k)|B|} = \frac{2k}{1+k} \tag{2.8}$$

In that case, $DSC(0.5) = \frac{2}{3}$, $DSC(\frac{1}{3}) = 0.5$ and $DSC(0.25) = 0.4$. These values, while showing the relative similarity level, are not very intuitive because they do not correspond to the actual similarity in a linear sense.

Another possible metric is the Jaccard index [190]. It is defined as

$$J = \frac{|A \cap B|}{|A \cup B|}. \tag{2.9}$$

It does not skew the values of similarity like Dice's score. Given the same A and B sets, the Jaccard index would be equal to

$$J = \frac{k|B|}{|B|} = k \tag{2.10}$$

However, if one set is not a subset of the other, this index considers their union. If the sets are highly similar, this score will enhance this similarity since the denominator will be smaller than in the Dice's score.

Despite these minor differences, both metrics are widely used in image segmentation evaluation. Indeed, Bertels et al. showed that there is no significant difference between the use of these metrics [191]. Cappabianco et al., however, pointed out that both metrics do not factor in true negatives [192]. Therefore, both metrics tend to be more sensitive to under-segmentation than over-segmentation. They provide an example where an image of 100k points is segmented with 0 false negatives and 20k false positives. It will have 0.83 Jaccard index and 0.91 Dice's coefficient, but 20k false negatives and 0 false positives will provide the scores of 0.80 and 0.89, respectively. In other cases, marking nothing in the image will correctly give both metric values of 0, however, the whole image marked as an object may yield large values in these scores, namely, 0.67 and 0.5.

2.3. Segmentation and clustering techniques

Segmentation and clustering are two distinct but related tasks. The goal of segmentation is to extract segments (usually separate objects) from an image, while clustering is diving the whole image into distinct clusters. However, segmentation is usually a task of supervised learning – it is solved by defining the objects and then finding them in the scene, while clustering is usually an unsupervised learning technique. While both problems can be solved by using machine learning, geometrical techniques also exist. They are all reviewed in this section.

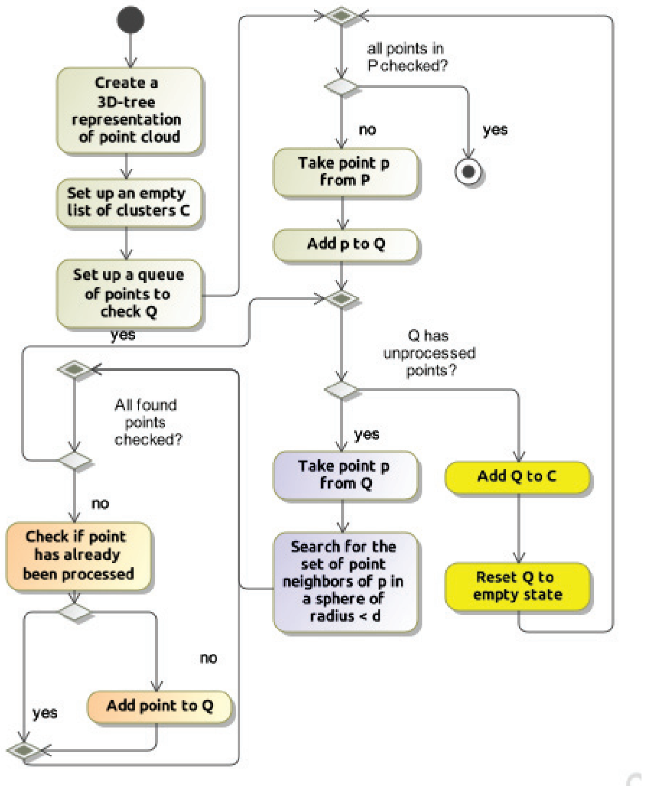
2.3.1. Geometrical segmentation and clustering techniques

2.3.1.1. Euclidean clustering

Euclidean clustering is an algorithm introduced by Rusu [49]. It is used to divide a point cloud represented by a three-dimensional tree into separate clusters. The algorithm is outlined as a UML activity diagram in Figure 2.1. It relies on the low logarithmic complexity of the search and runs it for each point, making the complexity of full point cloud clustering $O(n \log n)$. This algorithm has a hyperparameter d which is the radius in the radius search action. The radius search is an algorithm that, given a starting point p, finds all points with the Euclidean distance from p less than d. Euclidean clustering takes a point from a point cloud, finds all neighbor points for it, and repeats this search for each neighbor point until no new neighbor points are found. This collection of points is considered a cluster. When the cluster is finished, another unprocessed point is taken, and the process is repeated until all points have been assigned to a cluster. The purple color in Figure 2.1 shows Euclidean clustering actions, orange represents updating the search queue, and yellow indicates finalizing the cluster. This algorithm is included in a C++-based open-source library *Point Cloud Library*.

Euclidean cluster extraction has been utilized in different areas of research as one of the building blocks. The Treeseg algorithm uses Euclidean cluster extraction to split a point cloud into several smaller clusters [193]. The algorithm has also been applied in the rigid body pose estimation [194]. It was also used as a benchmark for SLAM reconstruction [195]. Euclidean cluster extraction was used directly to segment buildings and terrain of urban areas [196]. Euclidean cluster extraction was successfully sped up by Nguyen et al. by using GPUs [197], however, their solution uses a lot of memory, which makes it infeasible for large point clouds. On the other hand, this algorithm is not suitable for overlapping object segmentation [198].

Euclidean clustering utilizes the cluster itself as an object-defining feature. Another such feature is a bounding box, however, it is more commonly used for segmentation tasks than for clustering. It was used for collision detection [199]. The researchers implemented various techniques with a high-performance goal, one of them being the estimation of object bounding boxes. Another performance-oriented research tried to parallelize the detection of bounding boxes in a k-dimensional tree [151]. Bounding boxes are also often predicted by using neural networks. 3D-BoNet architecture was introduced to predict the bounding boxes from point clouds [200]. This architecture also emphasizes the efficiency of computation. YOLO3D was introduced as an extension to the existing YOLO v2 network and adapted to predict 3D bounding boxes [201]. This research is also no exception as it focuses on efficiency and real-time performance. Point clouds are also processed by extracting bounding boxes for 3D objects. Xu et al. implemented a neural network solution to estimate the bounding boxes [202]. They process both color data and point clouds to achieve better results. However, since this is a machine-learning solution, the researchers used publicly available datasets for training. Another solution for a similar problem was suggested by Zhou and Tuzel [203]. They implemented an end-to-end learning neural network architecture which processes depth data to get the bounding boxes. The researchers also used



Create a 3D-tree representation of point cloud
Set up an empty list of clusters C
Set up a queue of points to check Q
yes
all points in P checked?
no
yes
Take point p from P
Add p to Q
Q has unprocessed points?
yes
Take point p from Q
Search for the set of point neighbors of p in a sphere of radius < d
Add Q to C
Reset Q to empty state
All found points checked?
no
Check if point has already been processed
no
yes
Add point to Q

Figure 2.1. Euclidean clustering algorithm

publicly available datasets for training. A plethora of other neural networks have been proposed to find 3D bounding boxes for different types of objects: a point-level supervised network to find bounding boxes in lidar data [204], a fully convolutional neural network to find bounding boxes for vehicles in point clouds [205], or a graph neural network for point clouds [206]. However, all the presently mentioned methods are based on machine learning and use publicly available datasets for training. Most of them use the KITTI dataset [207] which was captured in 2013 and consists of 6 hours of traffic scenes. It is still widely used in current research because it takes a lot of time and effort to create a new dataset of that size. Even the bounding box annotation process, which is easier to perform for a human than segmentation masks, took 409 man-hours for 2806 aerial images with the 655k objects total to create the iSAID dataset, which is considerably newer than KITTI [208]. To sum up, it is evident that computing or predicting bounding boxes in depth data is a task that could be solved very efficiently. However, the state of the art does not reveal what accuracy benefits or drawbacks it introduces when compared to Euclidean cluster extraction, and machine learning solutions for this problem require hard-to-get datasets.

2.3.1.2. Bounding boxes in clustering

Bounding boxes are also used in clustering. However, the bounding box is usually used as a defining feature of an already-existing cluster instead of being required to find the clusters themselves. In other words, the object is found by using some method, and the bounding box is only constructed after that for a different reason. For example, the clustering method used for brain tumor detection in magnetic resonance images generates multiple candidate bounding boxes and then analyzes them by using a score function [209]. Bounding boxes are also used for deformable object tracking, however, they are merely used as the output of a whole data processing pipeline [210]. They are also applied in top-down detection methods [211, 156]. In contrast, Euclidean clustering is a bottom-up technique that starts from a single point and grows the cluster from there.

Bounding-box-based segmentation has its advantages as well as disadvantages. Some advantages of bounding-box-based segmentation include:

- Simplicity: The data model for the box is simple [212], and it is easy to represent it in memory – it requires a fixed (and small) amount of memory.
- Speed: Bounding-box-based segmentation is generally faster than more complex segmentation methods, as it only involves drawing a single box around the object rather than segmenting the entire image [213, 214].
- Flexibility: Bounding boxes can be easily resized or repositioned to better fit the object of interest, thus making it a flexible method for segmenting objects of various shapes and sizes [215, 216].

However, there are also some disadvantages to bounding-box-based segmentation:

- Inaccuracy: Bounding boxes may not always accurately enclose the entire object of interest, which would lead to incomplete or inaccurate segmentation [217].
- Limitations: Bounding boxes are not well-suited for segmenting objects with

complex shapes or for separating closely-spaced objects [218].

- Human intervention: Bounding-box-based segmentation typically requires manual annotation, which can be time-consuming and prone to human error [219].

Given these points, bounding boxes can be effectively used to segment non-touching separated objects in a scene. More complex scenes would require other means of Euclidean clustering and segmentation techniques. The applicability of bounding-box-based segmentation then depends on the use case. This research focuses on a single human segmentation from the background, therefore, the limitations are not as important. However, in cases where the human also touches another object, for example, sits on a chair, manual corrections to the segmentation output will be required.

2.3.1.3. Segmentation techniques

Segmentation is a related topic of research, however, instead of dividing the whole image into separate objects, specific classes of objects are found in an image. Related research shows that clustering techniques can be adapted for segmentation as well [220, 221, 222]. Euclidean clustering is no exception as researchers utilize it for segmentation purposes. It can be used to group points, thus essentially building a segment (or a cluster, depending on a point of view) [223] or splitting a segment into separate sub-clusters [224]. In all cases, the output must be interpreted further to decide which segment is the desired output.

An alternative method for object segmentation is the watershed [225]. This method considers an image as a topographic surface where pixel values can be imagined as height. The 'valleys' are filled with water until they start merging and the remaining 'hills' and 'valleys' are considered separate objects. This technique is used in combination with other techniques, for instance, the ResNet50 neural network [226] or K-Means clustering [227]. In the first case, the watershed method is used to find contour points which are then processed by the ResNet50 model. In the second case, it is applied to the output of K-Means clustering. However, watershed was shown to lead to over-segmentation with noisy images, local irregularities, and low contrast [228]. Watershed has recently been mostly applied to lidar data since it is natural to apply topological algorithms for such a type of data [229, 230, 231].

One of the challenges when working with watershed-based segmentation is the selection of the starting points. One possibility is to compute gradients over the image and find the local minima points. However, this usually leads to the too fine output of segmentation [232]. Alternatively, markers can be placed on the image [233]. This usually involves computing gradient magnitudes and reconstructing the minima and maxima in the image by applying dilation or erosion [234]. Finally, connected components analysis can be applied to find the markers [235]. The image is binarized, and then the connectedness of the components is analyzed. This step labels separated objects in the image, and these objects can then be used as the watershed initial minima.

2.3.2. Unsupervised clustering

The K-Means algorithm first defined by MacQueen [236] is a widely used unsupervised machine learning solution to segment point clouds. It has one hyperparameter

k (which is in the name of the algorithm) that defines the number of clusters to find in the point cloud (or other sets of points since it is not limited to point clouds). Initial points are selected as centroids, and each point is assigned to the segment of the closest centroid. Centroids are then recomputed to match the actual centroids of the segments, and the process is repeated until centroids converge, or another termination condition has been reached.

One of the issues with the K-Means algorithm is finding the optimal value of k if it is unknown beforehand. It matches the number of the expected segments. The elbow method has been suggested to find the best value of k [237]. It is based on the idea that the best amount of clusters is such that adding one more cluster does not give a lot of extra variance. Usually, the initial value of $k = 2$ is selected, clustering is performed, and the cost function is computed. k is increased with each iteration, and the cost is recomputed. The algorithm stops when the cost function value does not improve by a predefined margin with an increase of k. The cost function should be based on the variance of the clusters. Low added variance means that two clusters are next to each other, and k could be considered optimal. However, if the spatial deviation of the objects is low, the cost function changes will never be large, and the elbow algorithm will perform sub-optimally.

K-Means clustering is used in various segmentation tasks. Sauglam et al. achieved over 80% accuracy in building segmentation [238]. Zhou et al. also showed that the K-Means algorithm works best for background removal [239]. The algorithm was used for automated bridge deck detection [240]. The researchers used standard Matlab K-Means implementation. The algorithm may also be used together with other solutions. It was used with the YOLO3D network to improve its accuracy [241]. K-Means was applied to the output of YOLO-based detection and added a little accuracy (73.0% vs 72.8%) with a small performance cost (856 ms vs 848 ms).

These results suggest that K-Means could also be used to further segment the output of the existing segmentation algorithm since part of it may belong to the background.

2.3.3. Fully-manual segmentation

The segmentation of any image or other data is possible for a human given the right tool to mark the objects. Various studies show that this is the most reliable method of segmentation *for now*. A comparison between full-manual segmentation and two software solutions revealed that, while there is a correlation between the results, the software solutions do not outperform manual segmentation [242]. A different team of researchers also performed similar research with similar results [243]. They concluded that the analyzed software-based segmentation solutions overestimate the actual relevance of a medical image. Another analysis showed a similar picture – ICC values acquired from a software segmentation – are acceptable at best, not better, or at the same level [244]. Manual segmentation also was more accurate than machine learning models for prostate segmentation in MRI [245]. The accuracy of manual segmentation for human neurons is highlighted in a manual segmentation tool analysis [246]. The researchers state that automatic segmentation cannot yet compete with the segmentation performed by an expert. They show that the human segmentation accuracy ranges from 95% to 99% in this area of application.

On the other hand, this type of segmentation is prohitively time-consuming. Wild et al. suggested a 3D tool to mark contours in medical images [247]. It utilizes modern web technologies, and the user interface allows a highly visual segmentation output view. However, their research concludes that it takes about 35 minutes of manual work to create an accurate segmentation output. Other medical segmentation tasks can be even more time-consuming. For example, manual liver segmentation was estimated to take about 90 minutes [248]. A similar segmentation time was also reported by another study focusing on medical images for aortic walls [249]. However, this research also states that manual segmentation was 16% more accurate as well. Finally, a review of automatic segmentation techniques also highlights the issue of a large human time consumption for image segmentation [250]. The researchers also note that this is a routinely performed task. This shows that any time reduction would save a large amount of time over multiple segmentations. It is evident in a semi-automatic segmentation solution for MRI scans, which reduces the total time of segmentation from 479 to 167 seconds [251].

One of the issues with manual segmentation is the human bias. Research for manual segmentation of the prostate showed that there is a difference for some parts of the objects [252]. On the other hand, another medical manual segmentation research showed that the quality and other properties of the scan had more impact on the segmentation accuracy than the human bias [253].

The analyzed sources suggest that, while manual segmentation is the most accurate, it is highly time-consuming for a human. Therefore, even semi-automatic methods can save a large amount of time. This allows either more segmentation per human, or the saved time can be spent on other work. This also translates into financial savings if the segmentation is a paid activity.

2.3.4. Human body segmentation using neural networks

Neural networks is the most popular solution to segmentation tasks in the state of the art. Their main advantage is that they can extract features automatically. Handcrafted features are used in different segmentation techniques, however, this is a challenging task for the human, which has to be solved separately for each domain [254]. Neural networks and their working principles are reviewed in this section.

2.3.4.1. Image processing principles using neural networks

There are several neural-network-based approaches to segmentation tasks:

- Region-based methods;
- Encoder-decoder neural networks;
- U-shaped neural networks.

All techniques are a combination of a classifier and a segmenter – the features are extracted by a classifier and then processed by a segmenter. Therefore, segmentation CNNs are derived from classification CNNs.

One type of segmentation using neural networks the use of region-based methods.

They extract a region from an image and describe it with features. After that, a region classifier is trained. These data processing pipelines are complex as they are composed of multiple steps. One such method is the RCNN architecture [255], which proposes a three-stage pipeline. First, the regions are proposed by using selective search [256]. Then, a convolutional neural network processes the proposals and extracts features for the regions. Finally, a set of support vector machines is used to predict the object category. Another region-based architecture uses pre-processed images as the input for a convolutional neural network [257]. The network outputs features which are then interpreted by using a proposed algorithm. A variation of RCNN, Fast RCNN, introduces Region-of-Interest pooling, which is performed by analyzing regions of the output of the neural network [258]. Finally, the Mask RCNN architecture was proposed to segment instances rather than just detect objects [259]. The masks are extracted from the regions found in the image. The disadvantage of the region-based technique is the amount of work involved. A lot of regions have to be evaluated to correctly detect or segment the object in the image.

This problem is solved by combining fully-convolutional CNNs (CNNs with only convolutional layers) with a fully-connected network. The convolutional part of the architecture acts as a region-based feature extractor which is then processed (e.g., classified) by fully-connected layers. They provide the possibility to combine all activations of the previous layer. Therefore, they can extract global features which are useful for such tasks as image classification. This property makes the network trainable end-to-end for such tasks. A fully-connected layer is employed in a 3D CNN for brain lesion segmentation where it was shown to make predictions more structured [260]. The importance of a fully-connected layer is greater with shallower networks because the receptive field of an output neuron does not cover the whole input image. On the other hand, these layers add a lot of parameters to the network. For example, 89% of all parameters in the *VGG-16* network come from a fully-connected layer [27]. It was also shown that a fully-connected layer needs to have more nodes for shallower CNNs than for deeper CNNs [261]. In addition to that, convolutional and fully-connected layers provide different properties for the networks. Fully-convolutional architectures tend to achieve better translation invariance, which is useful for image classification – it does not matter where the object is localized in the image to determine its class. On the other hand, fully-connected layers add translation variance, which is useful for object localization tasks [262].

All types of convolutional neural networks are used to extract certain features from the original image. Usually, each successive layer has higher-level features than the previous layer. This is achieved by the non-linearity of the neural networks which is achieved by using such non-linear activation functions as SoftMax [263] or ReLU (rectified linear unit) [264]. Both activation functions were shown to be able to approximate any function up to arbitrary precision [265]. These functions, combined with the linear functions provided by the neurons themselves, approximate a function from a higher-level feature space than the previous layer.

Segmentation neural networks are usually derived from classification neural networks. A common architecture is an encoder-decoder network. The encoder processes data similarly to the classic fully convolutional networks – it extracts features from the original image. The decoder part then creates the image of segmentation masks from those features [266]. A common strategy in such networks is to add a skip connection – the decoder gets information not only from the previous layer but also from a layer of an encoder. These types of networks are called U-shaped CNNs or U-Nets. They were designed to bridge the semantic gap between the encoder and the decoder parts of the network [267]. This technique is widely used for many tasks like localization of image manipulations [268], computer tomography image analysis [269], image denoising [270], or cleaning image artifacts [271]. On the other hand, its effectiveness is disputed as it was shown that removing skip connections reduces the accuracy of segmentation by a small margin [272].

2.3.4.2. State-of-the-art neural networks for segmentation and classification

Tracking and separating human bodies from other objects requires a different approach compared to tracking and separating general objects because of the way the human body is shaped. Research has shown that the human body has a predictable geometric shape [273], unlike general objects which can have any shape [274]. This makes it harder for machines to recognize general objects compared to recognizing the human body. Therefore, segmentation using neural networks is usually domain-specific as one architecture to segment everything seems infeasible nowadays.

AlexNet is a type of neural network that was created in 2012 for image classification in the ImageNet competition [275]. It has 8 layers in total, with 5 convolutional layers and 3 fully-connected layers. Convolutional layers are used to process the image data and extract features, while fully-connected layers analyze these features and produce a final output. *AlexNet's* success was due in part to the availability of large datasets and improved GPU capabilities. Its performance demonstrated that convolutional neural networks could handle complex tasks. However, the use of fully-connected layers comes with a higher computational cost, as each neuron in the previous layer is evaluated for every neuron in the next layer [276]. *AlexNet* has many trainable parameters (62.3 million) due to fully-connected layers. *LeNet*, which was introduced in 1989, had only 0.4 million trainable parameters [277]. However, the hardware of the time was not capable of supporting larger networks. *LeNet* is still used today as a basis for other networks due to its light weight. For example, *LeNet-5*, a variant of *LeNet*, was used to detect COVID-19 from lung CT images [278]. It has only 82,000 trainable parameters but achieved an 86% classification accuracy.

The success of *AlexNet* inspired a lot of convolutional neural network research. *VGG-16* network was the next iteration of heavyweight network architectures [27]. It was proposed in 2014 and intended to explore the capabilities of very deep neural networks. This is even in the title of the original article – *Very deep convolutional networks for large-scale image recognition*. This was a fully convolutional neural network that utilized small 3×3 filters (*AlexNet* used much larger 11×11 filters). The network consists of 16 layers (hence the name of the network) that are grouped into 5 blocks. Each block consists of 2 or 3 convolutional layers with ReLU activation followed by a max-pooling layer. Each max-pooling reduces the dimensions of the previous layer by a factor of 2. The final layer is softmax to produce the final prediction. Since the network is very deep, it has a whopping number of parameters – 134M, which is two times more than *AlexNet*. This is one of the largest networks to date. It is still widely used in recent research. Examples include COVID-19 detection from X-Ray images [279], thyroid disease detection from cytological images [280], and lung segmentation [281]. The results presented in the mentioned research are very good, reaching an accuracy of over 97%. Research by Yu et al [282] has shown that *VGG-16* is better than *AlexNet* at removing background information. However, benchmarks performed by Canziani et al [283] have shown that *VGG* and *AlexNet* carry a small amount of accuracy per parameter. This is a disadvantage of these models as the researchers state that *VGG* and *AlexNet* are clearly oversized. *VGG-16* accuracy density was evaluated to be ~0.5% per million parameters, *AlexNet* was evaluated at 0.8% per million parameters. It was discovered that the parameter bottleneck is in the fully-connected layers and the network can be optimized by using dropout layers instead [281]. It was later shown that smaller architectures are possible and still provide reasonable accuracy [29].

The popularity of the *VGG-16* network inspired many other architectures. The *SegNet* neural network is one of the networks based on *VGG-16* [50]. It suggests an encoder-decoder architecture for semantic image segmentation — it takes RGB images as inputs and produces labels of semantic segmentation. Encoder-Decoder (ED) architectures are divided into two halves and are often referred to as U-nets in reference to the groundbreaking research by [267]. The spatial dimension is gradually decreased by the encoder using pooling layers, and the spatial dimension is gradually recovered by the decoder. By leveraging skip connections, each feature map in the decoder portion only gets data directly from the feature maps at the same level as the encoder part, thus enabling EDs to produce abstract hierarchical features with fine localization. *VGG-16* acts as an encoder in *SegNet*. The decoder is the reverse of the encoder — it has the same layers, but in the reverse order, and max pooling layers are replaced with upsampling layers. The authors introduced the idea of using pooling indices computed in the max-pooling step in the encoder. The network was trained to segment objects on the road into 11 classes. *SegNet* was also applied in other areas of research, such as brain tumor segmentation [284], detection of cracks in pavement [285], and semantic segmentation using event-based cameras [286]. *SegNet* was also shown to achieve low errors in ultrasound images [287]. This shows that *SegNet* is a very versatile architecture that can be applied in different areas. Mou et al also suggest a *VGG-16* encoder-decoder architecture based on *VGG-16* with relation modules [288]. They include spatial relation and channel relation modules that are then aggregated. They help to identify long-term relations in the images. However, *U-Net* is criticized due to the blurring of the extracted features and low-resolution information duplication [289]. *SegNet* is also a lighter architecture compared to *VGG* or *AlexNet* with 30M trainable parameters. The encoder-decoder architecture is popular for segmentation tasks as it has been used in numerous other solutions like TernausNet [290], DeepLabv3+ [291], a high-resolution multi-scale encoder-decoder network for blurry image segmentation [292], UNet++ for medical image segmentation [293] and many others. However, these networks are known to suffer from structural stereotypes and are difficult to train due to their depth [294]. The structural stereotype occurs when images are cut into smaller pieces due to hardware limitations, and then the labels are combined. This results in edge deterioration in predicted labels, and the edge area of the patches is incorrectly inferred. While *SegNet* is an encoder-decoder-based architecture, it was shown that a decoder is not required to achieve high accuracy in binary segmentation [295].

Segmenting the body posture is important, but using many features for a large dataset can take up a lot of memory. Some researchers have suggested using two different ways to reduce the number of features. They call this method "biview learning" [296]. They use two different views to show the differences in depth and the position between the body parts. This helps to simplify the features and use a support vector machine to analyze the data more easily.

An empirical study showed that CNNs *U-Net*, *SegNet*, *ResNet* and *Desenet* all perform with a similar accuracy for liver segmentation [297]. They all achieved Dice scores between 89.5% and 91.4%. *U-Net* and *SegNet* show a lower accuracy, however, *SegNet* has a simpler, more efficient architecture than other networks. Given the small difference between accuracies, *SegNet* seems to have the best accuracy and efficiency ratio.

Binary image segmentation is a sub-problem of segmentation where the image is segmented into the foreground and the background. It is a less-explored area of research because the goal is usually different from that for multi-class segmentation. The latter aims to understand the scene as each pixel usually is given a label. Binary segmentation, on the other hand, focuses on a specific type of object as it only segments the image into the foreground and the background. For example, given a scene with a human in a room, multi-class segmentation would try to label the human and each object in the room. It may find a wall, a floor, a chair, and a human. Binary segmentation, on the other hand, would find the human and label everything else as the background. Therefore, each binary segmentation solution is usually trained separately only with the type of data it has to recognize. An example of such an application is satellite image analysis which is solved via binary masks [298]. TernausNet v2 network is proposed to detect buildings in satellite images. Therefore, the goal of such a network is to learn the features of buildings as all other types of objects are not relevant in the area of application. Other examples include melanoma detection in images [299], brain tumor detection [300], or fire detection for embedded devices [301]. However, state-of-the-art solutions tend to use the same techniques for multi-class and binary segmentation. Even the already existing multi-class segmentation neural networks like *SegNet* can be adapted for binary segmentation efficiently [302].

There are other solutions for segmentation as well. A solution for binary image segmentation is SoftSeg [303]. The authors suggest that linear ReLU-based activation should be used instead of a sigmoid function to soften the boundary of the two classes. Spatial pyramid pooling was proposed by He et al. [304]. It produces a representation

Table 2.3. Overview of previous work on semantic image segmentation

Year	Model	Novelty	Major drawback
2012	AlexNet[275]	Depth of the model	Ineffective and lower accuracy than later models
2014	VGG-16[27]	Small receptive fields	Heavy model, computationally expensive
2015	U-Net[267]	Encoder-decoder architecture	Blurred features, slower due to decoder
2015	SPP-Net[304]	Variable image size adaptation	Cannot fine-tune convolutional layers before SPP layer
2015	FCNN[310]	Adaptation into fully convolutional networks	-
2016	ReSeg[306]	Recurrent layer	Features must be extracted by using other techniques
2017	SegNet[50]	Decoder non-linear upsampling	Slower due to decoder
2021	SoftSeg[303]	Normalized ReLU activation and regression loss function	Hard to evaluate due to fuzzy boundaries

Table 2.4. Comparison of deep learning model sizes

Model	Purpose	Parameters	Model file size
AlexNet [275]	RGB classification	62M	233 MB
VGG-16 [27]	RGB classification	134M	528 MB
SegNet [50]	Semantic RGB segmentation	32M	117 MB
U-Net [267]	RGB binary segmentation	30M	386 MB

of features that is independent of the input size. This is achieved by setting stride values of a pooling layer to be proportional to the input size. However, fine-tuning layers before the pyramid pooling layer is not possible with such an architecture. A hybrid of *U-Net* and *SegNet* neural networks was also shown to be a viable option [305]. The authors replaced deconvolution filters in *U-Net* with a SegNet-like decoder. It achieved up to 85% dice score in myocardium segmentation. A ReSeg model combined recurrent and convolutional neural networks [306]. This architecture has a classic encoder that produces its output which is then processed by recurrent layers. Another solution utilizing recurrent layers is DAG-RNN [307]. However, recurrent neural networks are not widely used for image segmentation.

Image sequences contain temporal information, however, sometimes the networks tend to learn more from other types of information, such as the background [308]. Some common challenges are present for object segmentation. They include object occlusion, deformation, motion blur, and scale variation. A related area of research is object tracking, which itself finds it difficult to track fast-motion, out-of-view situations. It was shown by a survey of video object segmentation and tracking that these problems are best solved by combined techniques [309].

An overview of the most important models presented in this overview is outlined in Tables 2.3 and 2.4.

2.4. Random forests

There is a set of classification problems where some rules have to be inferred to select an appropriate class. A common supervised machine-learning approach to this problem is the random forest [311]. It is an ensemble of some decision trees that are each trained with a different sub-sample of training data. This mechanism was shown to improve robustness and generalization while reducing overfitting, the properties sought-after in the machine learning community [312]. It has low training and, importantly, prediction times, has few parameters to tune, and, by nature, infers a set of rules for decision [313, 314]. They are used in different areas of application where such rules are possible to make [315].

Random forests can be applied to image segmentation as well. One of the related techniques is the graph cut. It is utilized in a plethora of applications where random forests learn the rules for graph cuts. It was used for image segmentation accuracy improvements [316], a random forest classifier for supervoxels/superpixels [317, 318, 319], and human limb segmentation in depth maps [320]. Random forests are used in segmentation by extracting some features from the image and training the classifier using those features. If the classifier works on the voxel basis, the possible features are the intensity, and the derivatives of the voxel and its neighbors [321]. However, the features seem to only be limited by the insight limitations of the researchers as it is necessary to find the features that carry information required to infer the classes. The features in state-of-the-art literature include projection ratios, eigenvalues of a covariance matrix, domain-specific features like elevation metrics, area and slope in airborne laser scanning [322], features learned by another machine learning method, such as a convolutional neural network [323], coordinates in standard space, gradient magnitudes [324], intensity [325] or a combination of learned and hand-crafted features [326]. These examples show that, generally, random forests can make good predictions if they are trained by using features that carry information to infer the desired classes.

2.5. Summary of literature overview

2.5.1. General findings

Kinect and *RealSense* sensors are the most popular among researchers, however, while *RealSense* devices offer a higher resolution, they tend to suffer from very high levels of noise. The comparison is outlined in Table 2.5. Results from related research suggest that *Kinect*, despite having a lower resolution, provides a more reliable depth stream, therefore, it makes more sense to use this device. While *Azure Kinect* has a lower level of noise, its non-rectangular field of view makes it more difficult to integrate it with state-of-the-art machine learning approaches efficiently. On the other hand, the noise levels in *Kinect 2* depth data are usually low enough and can be further reduced by using noise reduction techniques.

Depth maps and search trees both have their advantages and disadvantages. Therefore, they fit into different application areas. Depth maps are arrays of depth values, which makes it easy for applying the classic image noise reduction techniques.

Table 2.5. Comparison of popular depth sensors

Device	Kinect v2	RealSense D435	Azure Kinect
Depth resolution	512x424	1280x720	640x576
Noise	Lower	Higher	Lowest
Image shape	Rectangular	Rectangular	Hexagonal

Table 2.6. Comparison of depth data representations

Data representation	Depth map	3D binary tree
Easy to reduce noise	easy	difficult
2D search complexity	$O(1)$	$O(\log n)$
3D search complexity	$O(n)$	$O(\log n)$

Its depth values can be accessed in constant time if the 2D coordinates are known. However, searching for neighbors in 3D space is complicated because a large portion of the map has to be scanned. A 3D binary tree solves this problem. Both 2D and 3D search involves the same complexity for the binary search tree because the search is performed in the same way. On the other hand, noise reduction is more complex due to the higher search complexity in 2D space in comparison to depth maps. This dissertation focuses on two problems that requiring representations – automatic binary segmentation and semi-automatic binary segmentation. These can be solved in different ways, and both data structures may be useful. Geometrical methods depending on local search may benefit from the low 3D search complexity of a 3D binary search tree. On the other hand, convolutional neural networks are easier to use with an image-like structure, which, in our case, is a depth map.

Euclidean clustering is a widely popular algorithm. Bounding-box-based segmentation is often used because of its good performance, however, its effects on the accuracy have not been widely researched yet. Both algorithms have different properties, and they will be one of the focal points of this research. Related work may suggest that they might both be useful in different scenarios, which is going to be investigated in the following chapters.

Random forests were shown to apply to segmentation tasks, however, the means of application are different from the other reviewed methods. They operate on higher-level features that have to be provided to them. The usual approach is to extract features from the image either by applying predefined rules or via other machine learning solutions and then use the random forest to drop a part of an image. In the case of depth images, point clouds are usually used for this task. However, there are no universal rules on how to select the features as each research suggests its own features. Feature selection shall be explored further in this dissertation.

A bilateral filter is the best-suiting algorithm for segmentation because it preserves object edges best. However, other classic noise reduction algorithms like median or Gaussian filters are also used for this task. Since it was shown by related research that the *Kinect* depth stream has a moderate amount of noise, the effects of the most common noise reduction techniques shall be investigated further in this work.

K-dimensional tree construction is a complex task since it either involves a lot of computations, or the tree is not well-balanced. The median-of-three algorithm seems to offer a good balance between the tree construction performance and the balancedness of the resulting tree, therefore, it was decided to use this algorithm further in this dissertation.

Convolutional neural networks (CNNs) are the most popular solutions to image segmentation. Encoder-decoder and U-shaped architectures achieve the best accuracy levels for this task. The most popular CNNs achieve a similar accuracy for segmentation tasks, therefore, model efficiency should be the decisive factor for base model selection. *SegNet* is the most attractive candidate as it is denoted by a lower parameter count and an efficient architecture.

Full manual segmentation is an accurate but highly time-consuming segmentation method. Any time reduction allows either a larger amount of data processed or less work, which also means lower financial costs for paid segmentation. However, related research suggests that this kind of segmentation is the most accurate, and semi-automatic (human-supervised) segmentation methods are more accurate than fully-automatic segmentation while being less time-consuming than fully manual segmentation. This once again confirms the relevance of the proposed semi-automatic segmentation solutions.

These are the most relevant pieces of information for this dissertation:

- The *Microsoft Kinect 2* device offers the best combination of the depth image quality and compatibility with currently available depth processing algorithms;
- The bilateral filter seems to be the best noise reduction technique for segmentation tasks, however, this needs confirmation;
- The Euclidean clustering technique is a commonly used algorithm, widely used to process depth data;
- Encoder-decoder neural networks and their derivatives are the most successful types of architecture in the state of the art, and *SegNet* seems to provide the best accuracy and efficiency ratio;
- The *Microsoft Kinect's* skeletal data accuracy can be improved by using multiple sensors, however, the already existing skeleton transformation and fusion techniques either require a lot of computations, marker objects, or lack deeper investigation.
- Random forests and K-Means clustering are viable options for segmentation and cluster, however, their parameters have to be considered beforehand.

2.5.2. Motivation for algorithm selection

Euclidean clustering and watershed are widely popular and successful algorithms for clustering and segmentation. Both algorithms seem applicable for segmentation and are both investigated in this research. However, as shown in Section 4.2.3, watershed segmentation did not yield the expected results, therefore, this dissertation focuses on Euclidean clustering and the possibilities to apply it for segmentation purposes. This algorithm naturally fits the problem of depth segmentation as it works with a spatial data representation (a three-dimensional search tree) as it is not derived from other algorithms for RGB segmentation. The use of bounding boxes promises better algorithm performance, therefore, this research tries to combine existing Euclidean clustering and bounding-box-based segmentation ideas.

Conducted experiments, presented in Section 4.2.4, have shown that both existing and proposed solutions lead to under-segmentation. K-Means, an unsupervised clustering algorithm, has been shown to work well for background removal. Since under-segmentation means that the segmented object is the desired object with some part of the background, this algorithm seems to be a good fit. Therefore, this algorithm has been chosen to be used for further processing of the segmentation output.

The biggest challenge when applying the K-Means algorithm is selecting its parameter K. Given the high complexity of the analyzed scenes, creating rules for this selection is very difficult. Usually, when such a situation is encountered, machine-learning solutions are used. The best machine-learning approach for creating rules is the decision tree or the random forest. Since random forests tend to provide a higher accuracy and there is enough data to train them, it was selected instead of trying to create hand-crafted rules.

CNNs are, without much competition, currently the most popular and successful methods for RGB image segmentation. These networks can also be applied for depth segmentation since a depth map is an image-like data structure. This research also involves capturing and annotating a dataset of binary human body foreground-background masks. Given the dataset and the success of CNNs, it is reasonable to adapt the so-far-developed ideas of CNNs for depth data. This research, therefore, attempts to improve the already existing multi-class RGB segmentation CNN for binary depth segmentation. The *SegNet* neural network has been selected due to the following advantages:

- It is designed for high-resolution image segmentation – the analyzed images are of a higher resolution than most CNNs are designed to process;
- It is efficient compared to other popular CNN architectures for RGB segmentation – efficiency was one of its goals;
- It offers similar accuracy to other popular CNN architectures;
- Its variations have been shown to provide good accuracy for binary segmentation.

3. POINT CLOUD AND SKELETON DATA PROCESSING METHODOLOGY

This chapter presents the relevant theoretical grounds for the proposed algorithms as well as all other background relevant to this dissertation. First, a high-level process overview is provided in Section 3.1 as an introduction to the rest of the chapter. Next, a new metric for segmentation evaluation is introduced in Section 3.3. It is then used to evaluate all the proposed algorithms for segmentation in the further sections. Next, semi-automatic segmentation proposals are presented. Section 3.4 provides in-depth theoretical grounds and insights for a novel optimized bounding-box-based algorithm and compares it with a classic Euclidean clustering algorithm. This is a deep analysis that provides the overview of the required building blocks for semi-automatic segmentation, noise reduction analysis, the proposed bounding-box-based segmentation algorithm with three improvements and their theoretical performance and accuracy impact evaluation. It also describes how the algorithms can be extended for depth video sequences and how they can be adapted to work with large depth video files. Appendix C.1 is closely related as it explains how the algorithms are integrated into the software solution.

The proposed bounding-box-based semi-automatic segmentation algorithm was observed to suffer from under-segmentation errors. Section 3.4.3 proposes a machine-learning-based solution to reduce these errors.

Since a tool for semi-automatic segmentation was created, it has been possible to create a new dataset for a supervised machine learning architecture with a reasonable amount of human resources. *Agrast-6* architecture is proposed for this task, and its theoretical ground is presented in Section 3.5.

A skeleton transformation algorithm for multi-camera skeleton fusion is introduced in Section 3.7.

3.1. High level process overview

There are three main problems tackled in this dissertation:

1. How to assist the human in manual depth image segmentation to reduce the total time it takes to prepare a dataset for supervised neural networks;
2. How to automatically segment a human from depth images faster than when using state-of-the-art RGB-based semantic segmentation neural networks;
3. How to increase the accuracy of the *Kinect* skeletal data by using multiple devices.

The first two problems are related: to train a segmenting supervised neural network, a dataset is required. However, since creating such a dataset is extremely time-consuming [250], fast-performing assisting algorithms could be adopted for this task. Therefore, to both create a dataset and then train the neural network, the same data is processed twice – once with human supervision and then by the neural network which uses the output of the human-supervised segmentation. The last problem is only related to the *Kinect* device itself and focuses on accuracy improvements. The datasets

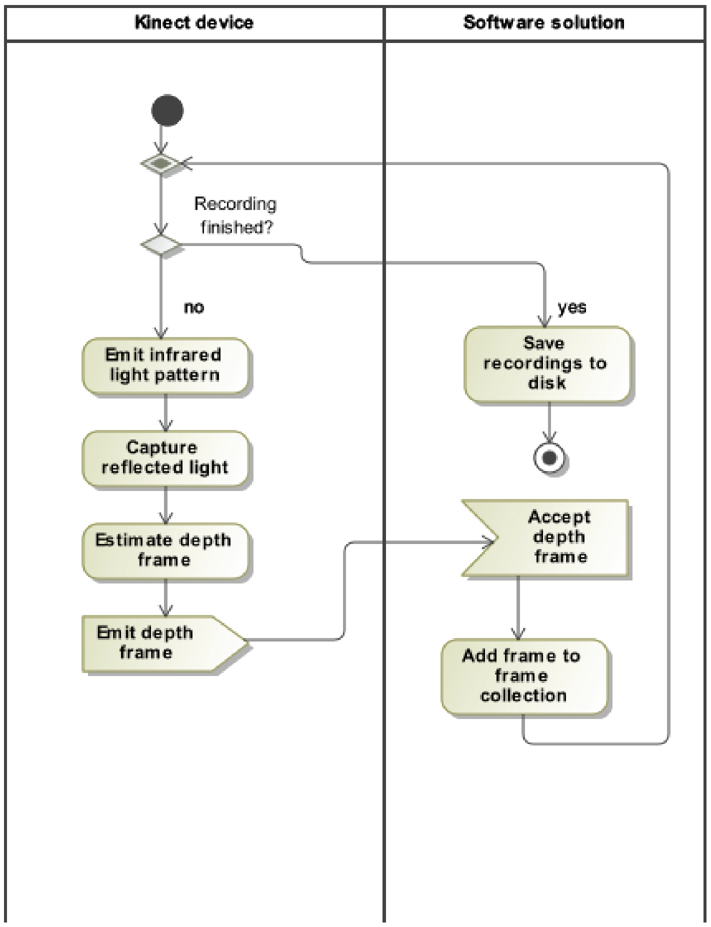


Figure 3.1. Recording capture UML activity diagram

captured for the first two problems utilized three *Kinect* sensors in the same way as required for the third problem, however, for a different reason.

The full process for setting up and then using automatic human segmentation:

1. A recording of depth data is captured by using custom software (the process from the device point of view is shown in Figure 3.1, and the process from the recording point of view is shown in Section 3.2, Figure 3.2).
2. Human-supervised semi-automatic segmentation is applied on the recording to get files of annotated recordings where human masks are marked on the frames (the process for one frame is shown in Section 3.4.1.5 in Figure 3.12, while the whole process is outlined in Figure 3.7).
3. A neural network is trained by using the produced labels and masks.
4. The neural network then gets depth frames from the sensor similarly to Figure 3.1, but, instead of just capturing depth data, it predicts and outputs human body masks that can be further used in a larger data processing pipeline.

The process for fusing the skeletal information from multiple *Kinect* devices is similar, however, instead of estimating and emitting depth frames, skeletal information is estimated and emitted by each *Kinect* device, collected to a single computer, and combined into a single, more accurate skeleton, and can then be used in a larger data processing pipeline.

This research focuses on the following details of the outlined processes and investigates them in-depth:

- High-performance semi-automatic depth data segmentation algorithm;
- Small neural network architecture (compared to state-of-the-art RGB semantic segmentation neural networks) for automatic depth data segmentation;
- Skeletal data fusion algorithm.

The fine details of each step of both processes are outlined in the following sections of this chapter.

3.2. Dataset collection

Two datasets have been collected for this research. Both datasets have been captured by using three *Kinect 2* cameras simultaneously. Special software has been implemented by using C# programming language to synchronize the recording of all sensors. Three computers were connected via a high-speed local network. One computer is considered the main piece, it has the user interface to start and stop recording. Other computers are instructed by the main computer – it sends a signal to the other computers to start and stop recording upon user interaction. The recording files are generated by using *Kinect Studio* API which is a part of *Kinect 2* SDK. XEF files are generated and saved on each computer locally with time stamps. The recording activity is outlined in Figure 3.2.

Recordings took place in closed environments with *Kinect* devices deployed around a person. Schematic deployment of the devices is shown in Figure 3.3. The sensors surround the person so that all the human's sides are visible. They are deployed at a height of approx. 1 meter, below the adult human chest level. The environments are indoors with a mix of natural and artificial light.

Both datasets were saved as XEF files, however, they can only be processed by using *Kinect Studio* as it is a proprietary file format. File sizes are large as well, 10-second recording takes 0.9-1.0 GB of disk space. *Kinect Studio* can capture a lot of information – RGB, depth, infrared, and other types of data. All the possible information was recorded during the recording sessions. However, only depth data was required for this research, which is only a small part of the whole data saved by the *Kinect Studio*. Another issue is that XEF files cannot be processed directly, and they have to be streamed by *Kinect Studio*, and then the stream can be consumed by other software. This is extremely inconvenient for research since the data is streamed at the same pace it was recorded. Therefore, custom software was implemented to convert XEF recordings into a different format. The converter is a custom-made software solution that captures depth frames provided by *Kinect Studio*. Each frame is added to the frame collection, and after all frames have been captured, the converter serializes the frames to a file. The frame consists of its dimensions and contents as a byte ar-

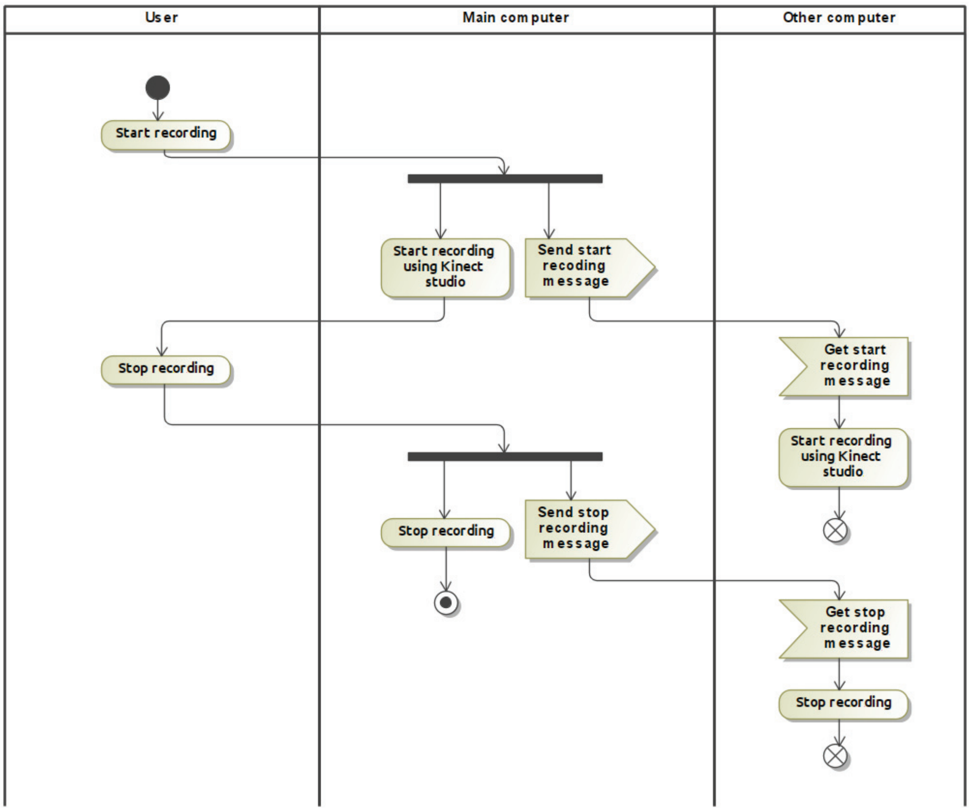

User
Main computer
Other computer
Start recording
Start recording using Kinect studio
Send start recoding message
Stop recording
Get start recording message
Start recording using Kinect studio
Stop recording
Send stop recording message
Get stop recording message
Stop recording

Figure 3.2. *Kinect 2* depth stream capture activity diagram

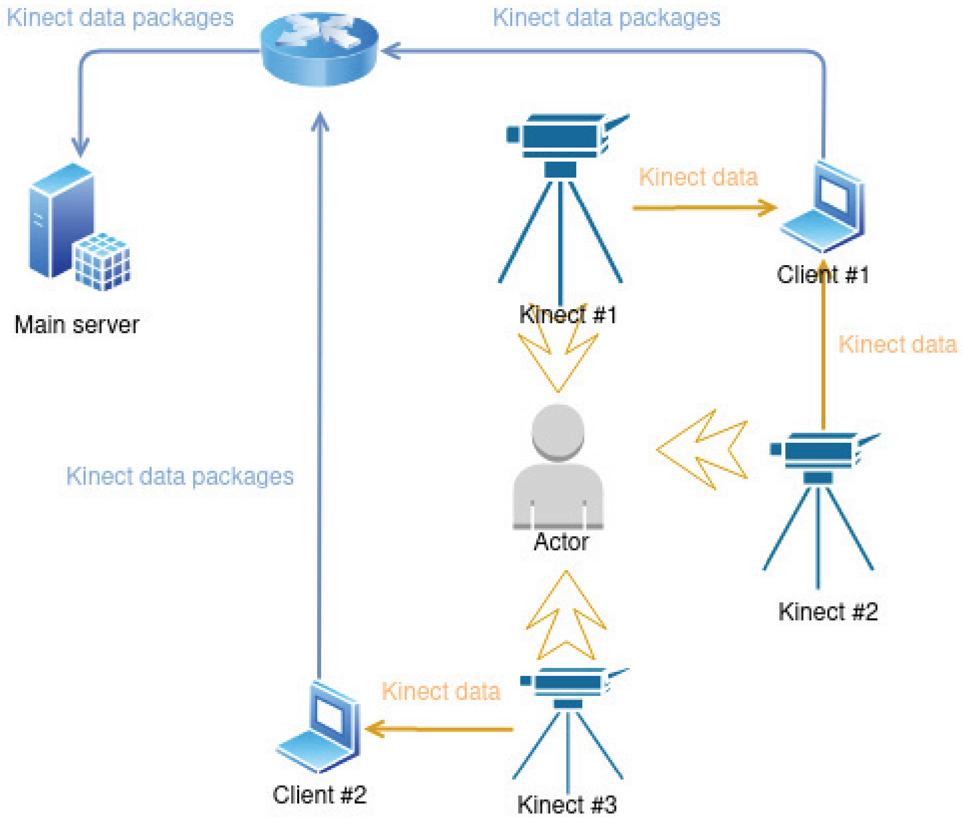


Figure 3.3. Deployment of *Kinect* devices during the data collection

ray. Depth maps were extracted from the recordings and serialized to files by using the *Protobuf* library. This reduced the size of the files to 160-190 MB per recording and made it possible to process data as needed as no streaming is required.

The datasets are stored as a set of files (one file per recording). They are used in combination with the semi-automatic segmentation solution. Binary masks are obtained with the combined efforts of the software and human supervision and stored in new files, again using the *Protobuf* library. The file consists of two video recordings – the original depth frame and the obtained binary mask. Since the depth frame is preprocessed by using a bilateral filter, the frame with this filter applied is saved. These files can then be loaded into a neural network training pipeline. Since the neural network proposed in this work learned binary masks from depth images, the two components are the input and the expected output of the network. If data is loaded directly from the *Kinect* sensor, it requires the bilateral filter beforehand for training or predicting the binary masks using the proposed neural network.

3.3. Cross-set intersection metric

Let us say we have two sets of points – G (ground truth) and A (segmented image). Then the cross-set intersection coefficient could be defined as

$$C = \frac{|A \cap G|^2}{|A||G|} \tag{3.1}$$

This metric introduces a non-linear relationship between the sets and their intersection. The square in the numerator originates from the idea to compute fractions of A and G that belong to this intersection. Formula (3.1) represents the product of those two fractions. Other metrics introduced in the literature overview are linear in this sense – they operate with values and their sums, not products. This metric is therefore stricter than the linear Dice's coefficient or the Jaccard index – if one of the fractions is low, the whole product will be reduced. In contrast, reducing the size of one part cannot reduce the overall score to a lower value than for the other components.

The cross-set intersection has an advantage over the Dice's coefficient because the metric carries an easily interpretable value when one set is a subsection of the other. This is important with semi-automatic segmentation in the case of under-segmentation. If the human body has been under-segmented (the whole body and some background have been captured), a manual correction is required. The size of the correction is then easy to estimate from the cross-set intersection – if the value is C, the area of the whole selection is C times larger than the actual ground truth. More formally, the amount of manual correction a is equal to

$$a = \frac{1}{C} - 1 \tag{3.2}$$

This property makes it easy to estimate a. However, this also holds for the Jaccard index since both metrics have the same values when one set is a subset of the other.

The cross-set intersection has an advantage over the Jaccard score as well. If two sets are given, their size evaluation is usually constant time operation. The intersection

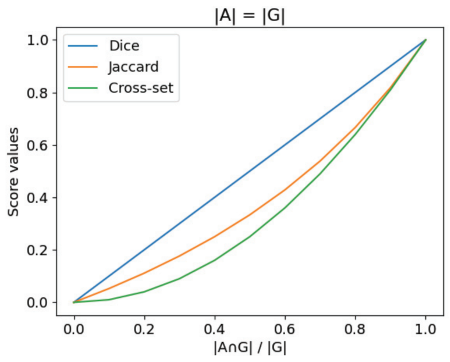


Figure 3.4. Metric comparison when $|A| = |G|$

is required for both the Jaccard score and the cross-set intersection, however, the Jaccard score also requires the size of the union of the sets which takes non-constant time to compute. Therefore, the cross-set intersection is guaranteed to be computed faster than the Jaccard score.

The cross-set intersection is denoted by one more interesting property. Since it is a product of part of the true positives in the predicted image and the true positives in the ground truth and both fractions are always in the range $[0; 1]$, it means that the cross-set intersection always fulfills the following criterion:

$$0 \leq C \leq \frac{|A \cap |G|}{|A|}, \frac{|A \cap G|}{|G|} \leq 1 \tag{3.3}$$

This means that the cross-set intersection is capped at the lower value of the two fractions. This property prevents relatively high scores if one of the errors is very large.

Let us consider three cases: where the ground truth and the predicted sets have the same size, the predicted frames are smaller (over-segmentation), and the predicted frames are larger (under-segmentation). Let us analyze these cases by using the Dice's code, the Jaccard index and the cross-set intersection metrics. In the case of the same frame size, all metrics yield the same result for 0 intersection and full intersection (the frames are equal). This is expected because all three metrics may have values from 0 to 1, and the edge cases represent the full prediction miss and the full prediction hit. However, when the intersection size is increased, the Dice's coefficient grows linearly since the denominator in Formula (2.7) is constant if the sizes of the sets are constant and the numerator is simply the intersection itself. The Jaccard score, on the other hand, is non-linear, while the cross-set intersection is parabolic. These results are outlined in Figure 3.4.

For the case of over-segmentation, zero intersection means that the prediction has no intersection with the ground truth, and the maximum intersection means that the prediction is a subset of the ground truth. Two cases are considered – the prediction is two times smaller than the ground truth and five times smaller than the ground truth. In this case, the Dice's score gives the best scores capping at 0.67 and 0.33, while the

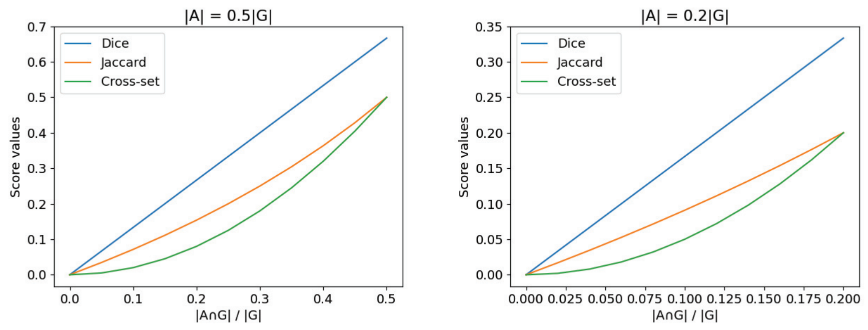


Figure 3.5. Metric comparison in case of over-segmentation

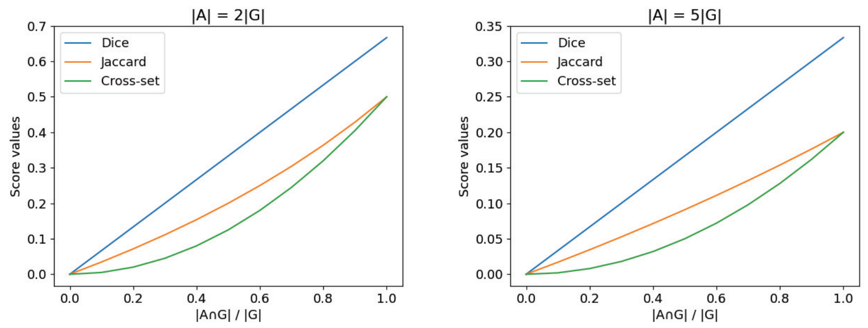


Figure 3.6. Metric comparison in case of under-segmentation

other two metrics cap at 0.5 and 0.2. Over-segmentation means that the segmenter missed some part of the object, however, the Dice's score is still relatively high. On the other hand, values of 0.5 and 0.2 carry some meaningful information that could be interpreted, and the part of the image that was captured successfully. However, this value is achieved differently for the Jaccard score and the cross-set intersection. In the case of the Jaccard score, this means that the intersection is 2 or 5 times smaller than the union. In the case of the cross-set intersection, it is a product of two values – the whole predicted image matches the ground truth (thus, the value is 1) and a fraction of 0.2 or 0.5 of the ground truth was predicted correctly. Therefore, this metric penalizes partial results more – if both parts are partially correct, the product of the partial results is smaller than for the linear metrics. The results are shown in Figure 3.5.

The metric behavior in the case of under-segmentation is similar to the results with over-segmentation. The Dice's score is again giving the highest scores where the Jaccard index and the cross-set intersection are stricter and easier to interpret. The results are visualized in Figure 3.6.

The analysis shows that all functions are monotonic with regard to the intersection size given fixed set sizes. This is a desirable property, and the cross-set intersection has it. However, in contrast to the other metrics, the suggested metric is stricter when the predicted set has both the true negatives and the true positives than the other metrics.

The cross-set intersection will give higher penalties to predictions that fail in multiple ways because it accounts for two types of errors, and if one of them is low, the total score will not exceed the lower score.

Since the cross-set intersection metric is quick to compute and is monotonic, the value in pure over-segmentation and under-segmentation is easy to interpret and use for manual work amount analysis, it will be used throughout the segmentation experiments.

3.4. Semi-automatic segmentation

3.4.1. Point cloud segmentation via bounding boxes

This section describes a novel proposed modification of an existing Euclidean clustering algorithm:

- The Euclidean clustering algorithm is adapted for semi-automatic segmentation in a conceptually novel way – instead of performing full clustering, segments are extracted from the point cloud directly;
- The key building block of the algorithm (the Euclidean radius search) is replaced by a less accurate and more performant piece (the bounding box search). The bounding boxes are applied in a novel way – they are used for segmentation directly instead of just describing the output of segmentation;
- The algorithm's internal state is moved inside the search tree, which enables existing improvements without modifying the structure of the search tree;
- The use of bounding boxes enables a novel improvement (an auto-expanding bounding box) which is only possible when the shape used for the search is defined by a fixed amount of values.

The following sections provide all the fine details of the problematics, adaptation of the algorithm and improvements.

3.4.1.1. Problem statement

Let us say we have a sequence of video frames (a depth video)

$$S = \{I_0, I_1, \ldots, I_n\} \tag{3.4}$$

Here, n is the number of frames (depth images) in the video. The goal is to find a binary mask for each frame that indicates the foreground and the background:

$$S_M = \{M_{I0}, M_{I1}, \ldots, M_{In}\} \tag{3.5}$$

Each mask in S_M is found from the corresponding frame in S. Therefore, the goal is to find such an algorithm that acts as function S_C from Formula (1.2). The algorithm should have a combination of performance and accuracy which reduces the computation time of computer-aided segmentation compared to the baseline PCL Euclidean segmentation as well as fully manual segmentation. Multiple variations of the algorithm may be viable for different scenarios and will also be considered.

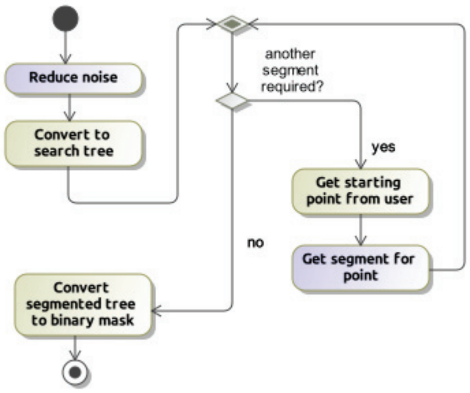

Reduce noise
Convert to search tree
another segment required?
yes
Get starting point from user
no
Get segment for point
Convert segmented tree to binary mask

Figure 3.7. Segmentation activity and relevant building blocks

3.4.1.2. Overview of required building blocks

Semi-automatic depth image segmentation can be imagined as a sequence of steps. One of the steps is performed by a human – he has to select a starting point that corresponds to parameter m in Formula (1.2). The other steps are performed by the segmentation algorithm and are outlined in Figure 3.7. The input for the activity is a depth map. First, the depth map is denoised. The denoised map is then converted into a binary search tree. When the human requests more segments, a starting point m is retrieved from the human, and a segment is constructed for this point. When the segmentation is done, the collected segments are converted into a binary mask which is the output of the activity.

The purple blocks in Figure 3.7 are considered in this section. Noise reduction and segmentation can be done by using different algorithms and are investigated in this research. Each solution has a different performance and a different impact on the segmentation accuracy. Therefore, based on the experimental results, guidelines for algorithm selection for purple actions shall be provided.

3.4.1.3. Noise reduction

The literature overview has shown that the bilateral filter is the best noise-reducing filter for the depth map segmentation. However, it has two sensitivity parameters σ_D and σ_R. The optimal values of these parameters depend on the data being denoised. In addition to that, it is more computationally expensive than the Gaussian or median filter. It is also common to apply a hole-filling algorithm for the depth data originating from the *Kinect* sensor.

3.4.1.4. Depth map segmentation using watershed

Watershed is originally a clustering algorithm. It could be adapted to segmentation tasks by clustering the scene and then selecting the correct cluster.

The clustering stage is performed by utilizing the watershed algorithm. The local minima are selected, and then water is filled from each minimum point. As the water level rises, the waters of different starting points meet. These meeting lines are considered object boundaries.

The data processing pipeline consists of image preprocessing, connected components estimation, and watershed segmentation. The image is smoothed. Connected components are estimated for the smoothed images and then the watershed segmentation is performed.

The next step is selecting the cluster that will be considered as the object. Since this algorithm is applied for semi-automatic segmentation, this can be picked by the supervising human. In other words, the watershed algorithm provides its output to the human, and the human selects which segments of the image should be considered as the foreground.

The advantage of this segmentation algorithm is that it processes depth maps directly. Since this data structure is the output of depth sensing devices, no conversions to other data structures are required, and standard programming tools for regular image processing can be used to implement this variant of watershed segmentation.

3.4.1.5. Point cloud segmentation using Euclidean metrics

Definition and theoretical analysis of semi-automatic and Euclidean segmentation

Extracting a segment from a point cloud while using the input from the human corresponds to Formula (1.2). It requires a parameter m from the user. Since general segmentation algorithms carry no semantic information about the objects being segmented, this parameter provides this information. It is required because, in the case of binary segmentation, only one object must be segmented from the whole scene. Say that there are a total of n distinct objects in the scene. Then there must be at least n values of parameter m that map to all possible objects in the scene. However, each object in the scene may be mapped to by an unlimited amount of parameter m values. More formally, given a scene with a set of distinct objects $S = S_1, S_2, \ldots, S_n$ in a depth image B, the segmentation parameter m_{ij} from the space of the possible values M must satisfy the following condition:

$$(\exists m_{ij} \in M) : (S_i = S_c(B, m_{ij}, h) \forall S_i \in S) \tag{3.6}$$

Since each object occupies a subset of the depth image, one of the ways to select parameter m is to use one of the pixels of the object as the defining feature. If it is assumed that one pixel cannot belong to two objects at once, i.e., all objects are non-intersecting, taking any point that belongs to the object will map the pixel to the object. More formally, we can use one of the pixels as parameter m if we assume that

$$\bigcap_{i=1}^{n} S_i = \emptyset \tag{3.7}$$

The nice property of selecting any pixel of the object is that it is generally very easy to determine by a human if they see the image. For example, when looking at the right image in Figure 4.1, it is quite easy for a human to tell where a person or a chair is located in the image. Therefore, a pixel from the object is a parameter that both fulfills the formal requirements and is easy to determine from an image for a human.

The segmentation can be performed by selecting a Euclidean metric to define the boundaries of the object. More generally, point cloud segmentation has to define the boundaries of the object. One of the ways to achieve this is to define a maximum value (a distance). This value is acquired by computing the distance from the object (this is defined by specific implementation of the segmentation) and a point. If the distance is computed as a Euclidean distance metric, the segmentation is considered Euclidean segmentation. If the distance is lower than a predefined value, the point is considered to be a part of the object. Since we have already defined parameter m to be one of the points of the object, this point is always a part of the object. However, after adding a new point to the object, the object itself is updated. Therefore, it must be accounted for when segmenting the object – the distance between a point and the object must be evaluated, but the object keeps changing during the process of segmentation. The segmentation is finished when the object converges.

One of the tasks is to determine the correct maximum distance. This depends on the data being analyzed and is represented by the hyperparameter h in Formulas (1.2) and (3.6). If the distance limit h is too low, this will lead to over-segmentation because the object boundary will be found too early. More formally, if the actual maximum distance between a point and the object during segmentation is h_{max} and $h < h_{max}$, this point will not be added to the object, which will lead to over-segmentation. On the other hand, if the distance between the object and the point that belongs to another object is $h_o < h$, this point will be falsely added to the object, and this will lead to under-segmentation. Therefore, the selection of h value directly impacts the quality of segmentation.

One more limitation arises for Euclidean segmentation. If there are points that belong to the object and have a smaller distance from the object during segmentation than a point that does not belong to the object, Euclidean segmentation will always either over-segment or under-segment the image. This can be inferred from the under-segmentation and over-segmentation definitions in the previous paragraph. A possible scenario is when both conditions hold:

$$h_o < h < h_{max} \tag{3.8}$$

This will lead to both under-segmentation and over-segmentation at the same time. Decreasing h may eliminate under-segmentation, but it will not solve over-segmentation or will even make it worse. Increasing h may eliminate over-segmentation, but it will not solve under-segmentation or will make it worse as well. Therefore, some cases are impossible to segment correctly by using Euclidean segmentation without data preprocessing.

Segmentation derived from Euclidean clustering

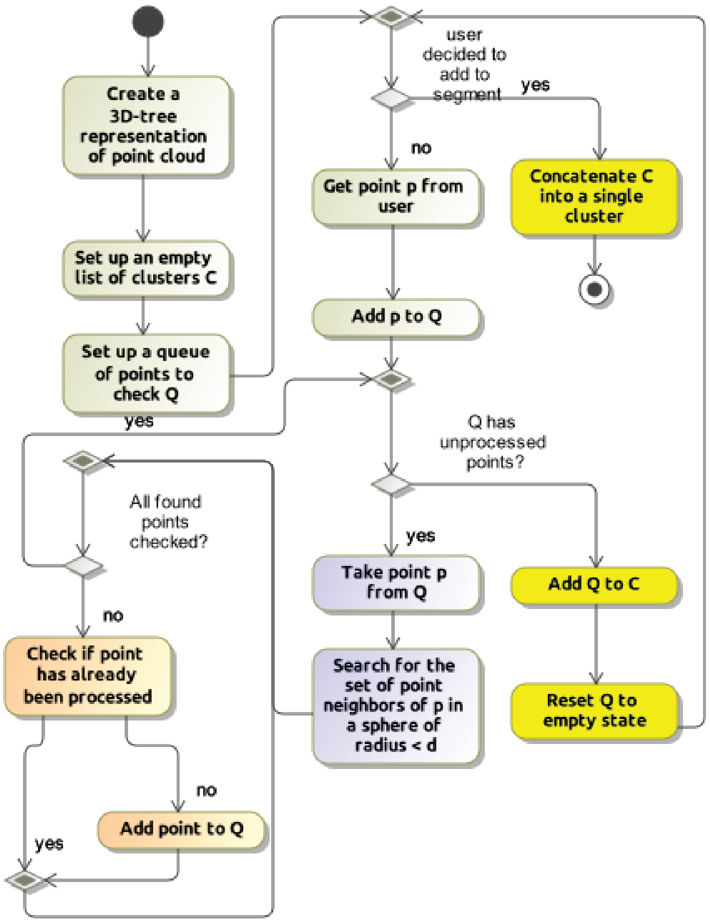
The Euclidean clustering algorithm introduced by Rusu [49] and presented in Figure 2.1 can be easily adapted for semi-automatic segmentation tasks. This variant of clustering uses the radius search – a small sphere with radius h is constructed around each point. If any of the spheres includes the analyzed point, it is added to the object. This solution treats the boundaries of the object to be a superset of all spheres around all points. Since the clustering starts by picking one point and, therefore, one sphere, and the point may only be added to the object if it is inside this sphere, this means that the object is fully-connected. In other words, for each sphere in the object, there is at least one sphere that it intersects with, and it is always possible to find a set of intersecting spheres that connect any two spheres. Segmentation is essentially a step of clustering – an initial point is selected, and a cluster is found for this point. This cluster can be considered a segment of the image, and if no further segments are constructed, this becomes a segmentation algorithm with a parameter – the initial point.

Radius search yields a complex shape of the object. If the object already consists of n points, the shape consists of n spheres. Each sphere is a separate object, and a containment check must be performed for each sphere separately. This means that the algorithmic complexity of the radius search is $O(n)$, where n is the current number of points.

Since the object is changing during segmentation, rejecting a point does not mean that it does not belong to the object. If there are 3 points p_1, p_2, p_3 and the distances between them are $D(p_1, p_2) = 2.5$, $D(p_2, p_3) = 2$, $D(p_1, p_3) = 2$, $h = 2.2$ and $m = p_1$, at first, the object consists of one point, p_1. p_2 is rejected because $D(p_1, p_2) > h$. Next, p_3 is added, which means that p_2 also has to be added. This property means that the radius search must be repeated for all added points to account for the whole object.

These two properties mean that the runtime of segmentation largely depends on the data. In the best-case scenario, all points are in the sphere of the initial point m, and no other points are added. If the total amount of points is k and the amount of points in the object is l, first we run a radius search around m. The complexity of this operation also depends on the data. It was shown by Lee and Wong that the worst-case range search complexity in a 3D search tree is $O(3n^{\frac{2}{3}})$ [327]. The best-case scenario is that all the required nodes are visited, and that no non-required nodes are visited, which yields the algorithmic complexity of $\Theta(k)$. Next, a radius search for all other found points must be performed. Since the number of points is k and the single search best-case scenario complexity is $\Omega(k)$, this gives the algorithmic complexity for the whole segmentation $\Omega(k^2)$. The worst-case scenario complexity, therefore, is $O(3kn^{\frac{2}{3}})$. For a *Kinect 2* depth frame with 217k points and an object of 20k pixels, this means at least 400k node traversals and at most 216M node traversals. However, both best and worst cases are not likely to happen with real data, therefore, the node traversal count could be expected to be in between those values.

Radius-search-based Euclidean clustering can be adapted for semi-automatic human-supervised segmentation. Figure 2.1 includes the action 'Take point p from P'. This action can be implemented as a user input – the user provides the point p manually, and the segmentation may then be performed. The figure shows the original segmentation scheme which segments the whole point cloud into different segments, however, in binary segmentation, only one object is required. It may be required to repeat the segmentation in the case of over-segmentation – if only part of the object is



Create a 3D-tree representation of point cloud
Set up an empty list of clusters C
Set up a queue of points to check Q
user decided to add to segment
yes
no
Get point p from user
Concatenate C into a single cluster
Add p to Q
Q has unprocessed points?
yes
All found points checked?
yes
Take point p from Q
Add Q to C
no
Check if point has already been processed
Search for the set of point neighbors of p in a sphere of radius < d
Reset Q to empty state
no
Add point to Q
yes

Figure 3.8. Euclidean clustering adapted to semi-automatic segmentation

segmented, the user may select another point, and the segment could be concatenated to an existing segment. Thus, the end of the algorithm is reached when the user decides on it. The adapted UML activity diagram is shown in Figure 3.8. The changes are in two actions – getting points from the user and cluster concatenation, and one decision – it is now performed by the user.

This kind of algorithm, while having suboptimal worst-case algorithmic complexity, provides fine-grained control of the object boundaries during segmentation. Each point must be in proximity (closer than h) to another point that belongs to the object to be accepted as a part of the object. This means that the algorithm may be applicable in some cases, especially in non-performance-critical applications, complex datasets, or where a higher accuracy is preferred over fast runtime. The downside remains the unsuitability to overlapping object segmentation [198].

Segmentation using bounding boxes

The main issue with the radius-search-based Euclidean segmentation is its performance. This problem arises from the fact that each point has to be checked for neighbors separately. This, in turn, is caused by the complex shape of the object defined by the spheres. Simplifying this shape could reduce the complexity of the segmentation.

To solve the search complexity problem, the object boundaries could be simplified. A neighbor search has to be performed for each point in the object because this is the only way to define the shape. This requires k tree traversals. The algorithm performance would increase if the number of traversals were reduced to $\log k$ or even to a constant time.

To have a lower search complexity, the shape should be defined globally for the whole object rather than combined from its parts. One of the most prevalent shapes in image processing is a bounding box. In the case of the classic 2D image, this is a rectangular box in the image. In a 3D world, this is a rectangular cuboid in the point cloud. It can be defined as different representations. The simplest representation is two points – the opposite corners of a rectangular cuboid. If an object consists of multiple points, the smallest possible bounding box could be considered an object boundary. This adds an extra assumption – the bounding box of the object that is being segmented should not intersect with the bounding boxes of other objects. More formally, the condition in Formula (3.7) must hold, but, this time, the bounding boxes of the objects have to be considered instead of the objects themselves. This may lead to a lower segmentation accuracy because using a bounding box instead of the spheres will tend to under-segment more often, especially when the environment is cluttered, or the object only takes a small proportion of the volume of its bounding box.

Using a bounding box for the segmentation requires a slightly different algorithm from the radius-search-based segmentation. There is no need to check a bounding box for each point since the bounding box is object-global. However, since the object changes with the search, this search must be performed multiple times because the bounding box may also have to be updated to fully contain the object. The algorithm is outlined in the UML activity diagram in Figure 3.9. First, an empty list of points is created. A bounding box for the initial point is created. Similarly to the radius-based search, parameter h is involved: the bounding box is increased by h in all directions. Thus, if the initial point has coordinates in 3D space (x, y, z), the bounding box opposite corners coordinates will be equal to

$$BB(p) = [(x - h, y - h, z - h); (x + h, y + h, z + h)] \tag{3.9}$$

Then, instead of performing a radius search, the bounding box search is performed. If some new points have been found, they are added to the list of the found points. The bounding box is then expanded to be the smallest possible bounding box for the point list, expanded by h. First, the minimum and maximum values for each coordinate are found across the point list. Then, the new bounding box is computed as

$$BB(L) = [(x_{min} - h, y_{min} - h, z_{min} - h); (x_{max} + h, y_{max} + h, z_{max} + h)] \tag{3.10}$$

This search-expand loop continues until the search converges. In other words,

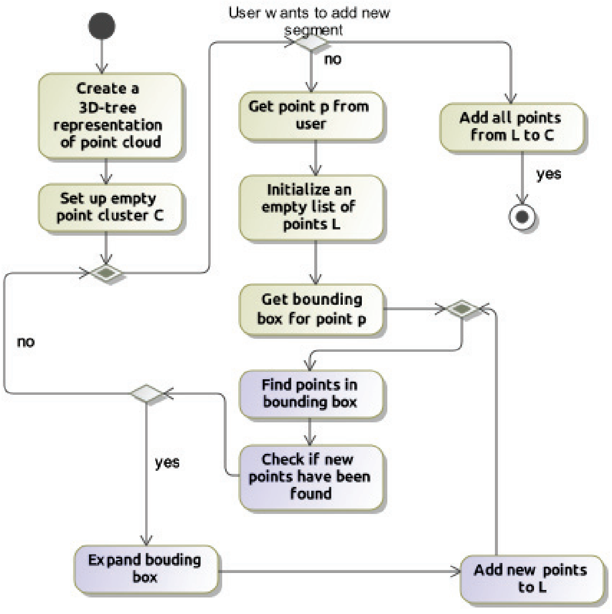

User w ants to add new segment
no
Create a 3D-tree representation of point cloud
Set up empty point cluster C
Get point p from user
Initialize an empty list of points L
Add all points from L to C
yes
Get bounding box for point p
no
Find points in bounding box
Check if new points have been found
yes
Expand bouding box
Add new points to L

Figure 3.9. Bounding-box-based Euclidean segmentation UML activity diagram

the smallest bounding box such that if expanded by h in any direction and contain the same subset of the point cloud as without the expansion for an initial point m is found and all the points contained in this bounding box are considered a single cluster.

This approach may save a lot of tree traversals compared to the radius search. In the best-case scenario, all points fit into the initial bounding box. Then, only 2 searches have to be performed – the first iteration collects all points, the bounding box is expanded, and the second iteration confirms that there are no new points to add. The bounding box search has the same complexity as the radius search since the algorithms are similar in their nature – the only difference is the containment check, which, in both cases, is a constant time operation. Therefore, the best-case scenario would require $\Omega(2k)$ iterations over the search tree if k is the number of points in the cluster and n is the total amount of points. The worst-case scenario complexity is the same as in the radius search, however, the data for this scenario has to be specially crafted – each bounding box expansion has to add exactly one new point until all points have been found. This is extremely unlikely when segmenting real localized objects, however, it is still possible for special cases. If the amount of bounding box expansions is $logk$, the complexity of the search becomes at least $\Omega(k \log k)$ and at worst $O(3 \log kn^{\frac{2}{3}})$, which in both cases will be much less than with the radius search. However, an average pass will likely be slower for a bounding box search since it will become much larger than the small sphere in the radius search, and a larger portion of the search tree will have to be traversed. Despite that, it is still expected to benefit more from the lower

Table 3.1. Comparison of search method algorithmic complexities

Search method	Best case complexity	Worst case complexity	Average complexity
Radius search	$\Omega(k^2)$	$O(3kn^{\frac{2}{3}})$	$\Theta(k \log n)$
Bounding-box search	$\Omega(2k)$	$O(3kn^{\frac{2}{3}})$	$\Theta(\log k \log n)$

number of searches which lose due to a slower single pass on average.

A summary of algorithmic complexities is shown in Table 3.1. It is assumed that both the radius search and the bounding box search will have $\Theta(\log n)$ complexity on average, and that the bounding box search will be performed $\log k$ times.

This proposal is a novel application of the bounding box. Usually, bounding boxes are used as a result of segmentation, however, the bounding box is just an intermediate representation of fast cluster growing, and the output is a subset of the point cloud. The bounding box is used as a prediction target in many neural-network-based segmentation methods [328, 329, 330] and other solutions presented in Section 2.3.1.

Bounding-box-based segmentation improvement 1: marking nodes as already removed

As discussed in the previous section, the bounding box search has to be performed multiple times with an increasing bounding box. This leads to the repetitive tree node traversal, and the same points are returned multiple times as a result of the bounding box search. This adds to the computation time. Figure 3.9 contains actions ‘find points in bounding box’ and ‘check if new points have been found’. These actions perform more computations that are required to collect all the points contained in the bounding boxes.

The action ‘find points in bounding box’ is repeated in a loop. Since all points found in the bounding box must be returned as a result, the type of the result, in the Java type system, is `List<Point>` (see Figure 4.2). If the segmentation output consists of k points, the points have to be added to the result list k times during the search. However, the search is repeated multiple times with an expanding bounding box. Bounding box expansion, as defined in Formula (3.10), leads to the following property:

$$BB(S_1) \subseteq BB(S_2) \leftarrow S_1 \subseteq S_2 \tag{3.11}$$

In other words, if one set of points is a proper subset or equal to another set of points, its bounding box will be smaller than or equal to the bounding box of the second set. The condition $S_1 \subseteq S_2$ will always hold during the iterations of the bounding-box-based search, therefore, if the points were included in one iteration, they are guaranteed to be included in all subsequent iterations. This leads to the fact that these points will be added to the result list multiple times, however, it would be enough to only add them once, which would reduce the amount of work done by the search algorithm.

The action ‘check if new points have been found’ also suffers in terms of performance for the same reasons. Two sets must be kept in memory and compared – if the sizes are the same, no new points have been found, and vice versa.

This problem cannot be solved without any modifications to the search algorithm. It is not enough to only have the search tree because it carries no information on whether the node has already been found in a previous iteration. This leaves two options – either we add a parameter that carries information about the already found points, or we add an extra state to the tree itself.

The first option involves adding the list or a set of the already collected points as a parameter to the search function. This would then involve the case if the point is already among the collected points, and if yes, it could be skipped. However, this search would have to be performed for each analyzed point. While hash sets have a low containment check complexity, their creation requires extra work, and the hash function must be highly efficient. This means that extra development work needs to be done to have a well-balanced hash set, and it would still decrease the performance because the set has to be created in the first place.

The second option does not have any of the downsides that the first option suffers from. One of the ways to add a state to the search tree is to have a Boolean flag for each node. The flag would indicate whether the point has already been collected in a previous iteration or not. The Boolean flag check is faster than any containment in any data structure and requires no hash computation. Its downside is that the Boolean flag is required for every node in the tree, which increases the memory consumption. This solution has another implication – if the search tree has to be reused for a new search, its state has to be reset, which is an $O(n)$ complexity operation – it is required to traverse all nodes and update their flags.

The introduction of improvements has implications on the implementation of the algorithm. When no improvements are used, all the state of the algorithm must be tracked outside of the search tree. The algorithm starts with an empty cluster and fills it with points as the algorithm progresses. It has to keep track of the processed and unprocessed points to know which point has to be processed next, which points should be discarded, etc. The implementation uses `HashSet` to effectively track the found and processed points, however, the amount of the required set updates and containment checks is very large because the same points can be found during different passes over the tree. The proposed improvements do not need this type of state tracking as the state is stored inside the search tree – `the removed` flag has to be changed instead. This flag is then checked for every node in the tree during the passes over the search tree.

The internal workings of the improvements can be visualized by using a one-dimensional search tree as shown in Figure 3.10[1]. In the case of one dimension, a bounding box search is just a range search. In Iteration One, the search range is $[27; 32]$. The root node is 50, so it is only required to check the left branch. 25 is found, the right branch is selected. Then, the whole branch must be checked because 30 is in the search range, and the values of 28 and 30 are marked as removed. The bounding box is expanded to $[25; 32]$ because 28 was close to the end of the range. Iteration Two is similar, except that it checks a wider range, and the nodes with values 28 and 30 are not checked, they are skipped, and only their children are examined.

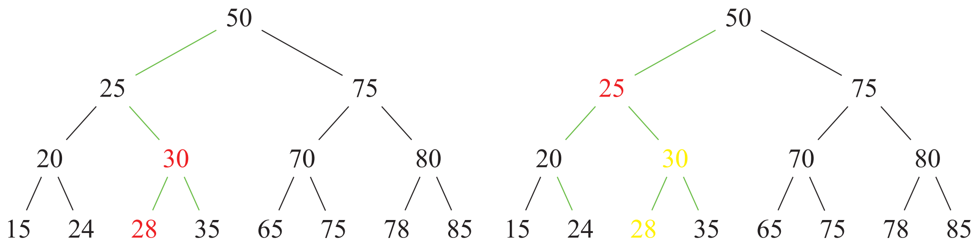


Figure 3.10. Marking nodes as removed. Left: iteration 1 ([27-32]), right: iteration 2 ([25-32]). Red – found nodes, green – search path, yellow – skipped nodes

Given the advantages and disadvantages of both solutions, adding a state directly to the tree promises more performance gains at a cost of some extra memory consumption. This seems like a trade-off that is worth making, therefore, it has been chosen for the final implementation.

The performance impact of these improvements, however, largely depends on the data and the implementation itself. Let us consider three scenarios and analyze the impact of these improvements in those scenarios. All scenarios will assume the point cloud size $n = 217.088$ and the cluster size $k = 20.000$. Only the result list collection will be considered because the other parts of the algorithm remain the same.

The first scenario is where all points are found in one go. This requires two iterations over the search tree – first to find all points, and second to confirm that the expanded bounding box adds no new points. Without the improvement, this would mean that the resulting list of points is the same for both iterations, and that size is k. This means $O(2k)$ complexity of filling the list with the found items. With the improvement implemented, this would decrease to $O(k)$ because the first iteration would be the same, but the second one would find no new points, which means that no additions to the list will be required.

The second scenario is where $\log k$ iterations are required to find all points, with each iteration adding $\frac{k}{\log k}$ new points. Without the improvement, this would mean that the list will have to be appended a total of $\Sigma_{i=1}^{\log k} i \frac{k}{\log k}$ times. Given the assumed scenario, this would result in about 92k list append operations. With the improvement enabled, this would stay at $O(k)$, which is 20k list append operations.

The third scenario, which is the worst-case scenario, is where only one point is added in each iteration. Without the improvement, this would mean that there are k iterations required, and the list size in iteration i is equal to i. Therefore, the total amount of list append operations is $\Sigma_{i=1}^{k} i$. This is an arithmetic progression, which means that the sum can be rewritten as $k\frac{1+k}{2}$. For $k = 20.000$, this means about 200M appends. Without the improvement, only one point will be appended per iteration, which again yields $O(k)$ appends.

A summary of the performance impact on list append operations is provided in Table 3.2.

[1]Proof that trees grow from the top is presented in Figure A.1 in Appendix A.

Table 3.2. Comparison of number of list append operations

Optimization enabled	Yes	No
Best case complexity[2]	$O(k)$	$O(2k)$
Average case complexity	$O(k)$	$\Sigma_{i=1}^{\log k} i \frac{k}{\log k}$
Worst case complexity	$O(k)$	$k\frac{1+k}{2}$

Another problem with a trivial bounding-box-based algorithm, as well as the Euclidean-clustering-based algorithm, is segment concatenation. If a segment is extracted from the point cloud and the user decides that another segment should be added as well, the second segment may intersect with the first one. The segments have to be concatenated, however, duplicates have to be removed. This essentially means that a set union has to be found. Another possible application is the full segmentation of the point cloud. In that case, if some points already belong to one segment, they cannot be added to another segment. This essentially means that a set difference has to be found. In both cases, extra work has to be performed to be able to work with multiple segments. To make matters worse, if one segment is already constructed and a second segment is being built, the second segment has no reason to include points already existing in the second segment. These problems can also be prevented by the suggested improvement. The nodes are marked as removed and will not be considered for other segments.

However, there is a possible difference in the output of segmentation in the case of bounding boxes. If one segment is already constructed and another segment is being constructed, some points may be omitted as already removed. This omission may lead to a different-sized bounding box which, in turn, may lead to a different segmentation output. However, whether this is an advantage or disadvantage, it depends on the use case. The research focuses on semi-automatic human body segmentation. In this case, if one segment, for example, the human torso, has been segmented, and the user selects an arm, it is more intuitive to select only the arm and the body part up to the torso, but not a leg which is on the other side of the torso. Therefore, this property of the removed point improvement will be viewed as an advantage in the scope of this research.

Bounding-box-based segmentation improvement 2: skipping fully removed branches

The first improvement, as defined in the previous subsection, solves problems of too many containment checks and list append operations. However, the search still performs more actions than required in some cases. If a tree branch already belongs to an already found segment, there is no point to check the nodes again – they can be skipped altogether. Even though this is a cheap operation, some time savings are still possible, especially for large branches. Due to the nature of the search trees, the nodes under the same root node are more likely to be spatially closer than under different nodes. Therefore, it is more likely that the nodes that belong to the same branch are

[2] As a fun note, complexity of $O(k)$ is actually OK in this context

going to be included in the same segment.

The exact node traversal reduction effect is difficult to measure, and it heavily depends on the data. A balanced three-dimensional search tree is constructed by selecting a pivot and putting all points lower than the pivot to the left branch, and the points larger than the pivot to the right branch. The pivot selection is a compromise between the tree creation performance and how well-balanced the tree is. If the pivot is the median value, the tree will be balanced. However, finding the median is not a trivial operation in an unsorted collection – it requires sorting the collection, which usually has $O(n \log n)$ complexity. An implementation suggested by Gashler [331] utilizes a three-median approach which was first introduced by Kirschenhofer [149]. This yields a nearly balanced search tree, so let us assume the balanced search tree search complexity.

The improvement will yield the best results with larger subtrees removed during the search. The probability of this happening is very difficult to estimate for real-life data. It depends on the object, the selected pivots during the tree construction, the initial point m provided by the user, the bounding box expansion parameter h, and the surroundings of the object. The segmentation starting from points m_1 and m_2 may lead to the same segmentation result, but the path to achieve it may be wildly different. This means that even if the object is conveniently placed in its subtree inside the search tree, m_1 may immediately mark the whole branch as removed, while m_2 will collect points from different branches first, and it will not be possible to rule out a large subtree until the final iterations of the search. On the other hand, if the same object is in a different environment, its points may be distributed in the search tree in different ways. Even the same scene can be represented by different search trees if the pivot is selected randomly. Since the nature of k-dimensional trees is that unpredictable, analytical analysis of this improvement is skipped, and only experimental results will be provided in the next chapter.

The improvement can be implemented in one of two possible ways, similar to marking points as removed. This information can be stored as the internal or external state of the search tree. If the external state is used, it needs to have a collection of already removed branches. However, collecting only branches is non-trivial because the branch is considered removed when the root of the branch as well as both child sub-branches are fully removed. Thus, if a node is removed, but its children are not, the branch is not removed. However, it has to be tracked which children are removed, and if a smaller subtree has been removed and its parent also got fully removed, the child also should be removed from the collection of the removed branches. The alternative is to just check the list of the already collected points, but this is a computationally expensive operation if the list of points gets large. Another alternative is to also keep a hash set of the points, but this consumes more memory, and the hash function has to yield few collisions.

Keeping this information in an internal state of the tree is, again, simpler. Each node only has to track its two children – if the node has been removed and both children are fully removed, the node is also considered fully removed. Leaf nodes are

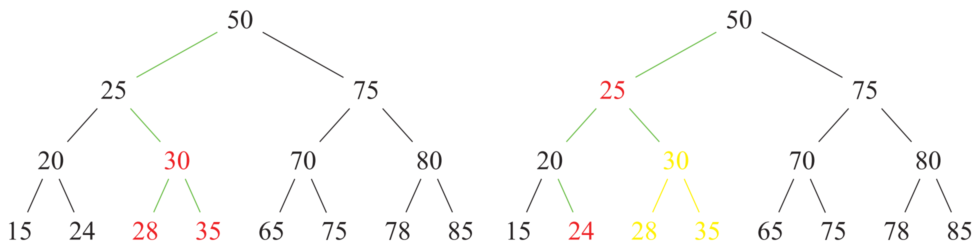


Figure 3.11. Marking subtrees as removed. Left: iteration 1 ([26-36]), right: iteration 2 ([24-36]). Red – found nodes, green – search path, yellow – skipped nodes and paths

fully removed when they are removed by themselves. This is trivial to implement for recursive data structures. When running the search, if a child is fully removed, it is not traversed at all since no new information will be found anyway. This, again, adds one Boolean flag to the search tree node – it indicates whether the branch that this node is a root of is fully collected or not.

This improvement can only be used together with marking points as removed. Otherwise, there is no way to know if the whole tree is removed without actually traversing it, which defeats the purpose of this improvement in the first place.

The internal working of the algorithm is depicted in Figure 3.11. During the first iteration, a bounding box $[24–36]$ is used. This finds the values 28, 30, and 35. Conveniently, they all make up a whole subtree. Nodes 28 and 35 are marked as fully removed because they have no child nodes and are removed by themselves. Node 30, in turn, is marked as fully removed because all of its children are marked as fully removed as well as the node itself. During the second iteration, node 30 is reached again, but the search immediately terminates for that branch since it is already known to be fully removed. As a result, three nodes and two edges are completely skipped during the second iteration.

This extra improvement is guaranteed not to change the output of the segmentation in comparison to only having the first improvement enabled. The nodes that are skipped are already removed anyway, so this improvement simply skips the parts of the tree that are guaranteed to yield no new points.

Bounding-box-based segmentation improvement 3: auto-expanding bounding box

The bounding box search is based on the idea that an initial small bounding box will expand and converge to the bounding of a distinct object. However, given the bounding box expansion h, if the total size of an object in one of the tree dimensions is H, at least $\frac{H}{h}$ bounding box expansions are required. Thus, since the complexity of the tree search may reach $O(3kn^{\frac{2}{3}})$, the total complexity of the search is

$$O(3\frac{H}{h}kn^{\frac{2}{3}}) \tag{3.12}$$

If H is much larger than h, many bounding box expansions are required to capture the object. However, Formula (3.12) shows the best-case scenario of expansion where the bounding box is expanded by h or a very close value. Unfortunately, this is not guaranteed, and the number of expansions may be even larger. In the worst-case scenario, each expansion may only add one point making the total complexity

$$O(3k^2n^{\frac{2}{3}}) \tag{3.13}$$

However, an improvement is possible. Figure 3.9 shows the bounding box expansion after running the search. This is not the only possible algorithm. What if we expanded the bounding box not after the search, but *during* the search?

In a general sense, the search shape is a parameter of the search. This parameter is recalculated and passed again in the next iteration. In the case of a bounding box search, the box size in the next iteration is defined by Expression (3.10). Values x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max} can be defined in terms of the found points. Given a set of found points $S = p_1, p_2, \ldots, p_n$, an extremum x_{min} can be defined as

$$x_{min} = \text{min}(p_{1x}, p_2x, \ldots, p_{nx}) \tag{3.14}$$

Formula (3.14) can be rewritten as

$$x_{min} = \text{min}(p_{1x}, (min(p_{2x}, \ldots), p_{nx})) \tag{3.15}$$

This means that the local minima may be computed in steps and combined into a global minimum. If we define the minimum of the first i elements of the sequence as M_i, it can be expressed as

$$M_i = \text{min}(M_{i-1}, p_i) \tag{3.16}$$

Formula (3.16) essentially means that the minimum can either be computed for the whole sequence at once, or by iteratively updating the minimum value for each point. Both versions of calculations will yield the same minimum value. The same holds for all other minimum and maximum values.

Formula (3.16) can be utilized for optimizing the bounding box expansions. Since it has already been shown that the global extremum is equal to the iterative extremum over the whole set, it means that, if the bounding box is expanded with each found point, its size will be the same as when expanding it by finding the extremum over the whole segment if the segment is the same. However, this allows for an extra feature. If the bounding box is expanded during the search, it means that an expansion of $\frac{H}{h}$ per iteration is not the limit since a larger bounding box may include extra points. This means that the complexity defined in Formulas (3.12) and (3.13) would be improved.

The introduction of auto-expanding bounding boxes alters the activity diagram defined in Figure 3.9. The action ‘expand bounding box’ is no longer performed after finding all points. Therefore, it is removed and merged with the action ‘find points in bounding box’. The updated UML activity diagram is shown in Figure 3.12.

While this approach expands the bounding box to a greater extent per iteration, it

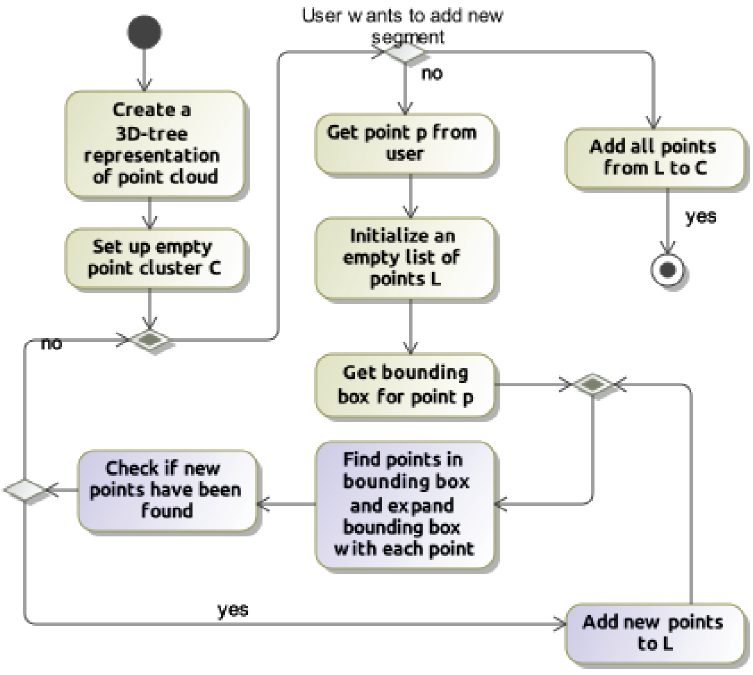

User w ants to add new segment
no
Create a 3D-tree representation of point cloud
Set up empty point cluster C
Get point p from user
Initialize an empty list of points L
Get bounding box for point p
Add all points from L to C
yes
no
Check if new points have been found
Find points in bounding box and expand bounding box with each point
yes
Add new points to L

Figure 3.12. Auto-expanding bounding-box-based Euclidean segmentation UML activity diagram

is not guaranteed to collect all the points that belong to the bounding box computed at the end of an iteration. If the initial bounding box size in one dimension at the start of iteration is H_1, in the middle of the iteration it may be $H_1+\Delta h_1$, and, at the end, it may be $H_1 + \Delta h_2$, where $\Delta h_1 \leq \Delta h_2$. If a point at the distance $\Delta h_1 < h_d < \Delta h_2$ from the center of the bounding box in the selected dimension is traversed before reaching the middle of the iteration, it will be skipped. Moreover, the whole branch may be prematurely cut during an early stage of iteration because it is unknown how large the bounding box is going to get.

The disadvantages may be well worth it though. The problems with missing some points during the search may be overcome by simply running the bounding box search one more time. Since the box cannot shrink, the second iteration is guaranteed to collect all points skipped in the first iteration. In addition to that, this second iteration is required anyway to confirm that the bounding box does not expand any further. This means that if the bounding box search is run iteratively until the bounding box has converged, this disadvantage becomes an advantage. Having the full-sized bounding box in the first iteration is impossible if the size of the object is larger than the bounding box increment h, which is true for all real-life applications – h should be on the scale of point cloud resolution, not the whole objects. This means that many iterations are required to have the full object. Let us analyze the case with one expansion for the sake of simplicity. Without the auto-expanding bounding box, three iterations are required – the first iteration traverses a part of the tree; the second iteration traverses a larger part of the tree which fully includes all nodes from the first iteration (given that the improvements from the previous sections have not been implemented). Introducing the auto-expanding bounding box may lead to a case where Iteration One yields the full-sized bounding box, however, a full subset of the tree for this bounding box may not be traversed depending on the order of traversal. The second iteration is the same as the third iteration without an auto-expanding bounding box. This means that the first and the second iterations are replaced with one iteration with a traversal count between those two iterations, and the third iteration remains as the second iteration with the auto-expanding bounding box. This scenario is not guaranteed, but the example is oversimplified. If many bounding box expansions are required, the probability of saving some iterations increases.

In the best-case scenario, the bounding box grows with every point in such a way that all points are collected in a single pass. The second iteration is then only required to check that no points have been missed. This was described as the worst-case scenario for a traditional Euclidean bounding box search with k iterations. Introducing auto-expanding bounding boxes leaves only 2 iterations in this case. The worst-case scenario is where expanding the bounding box adds one point but misses other points due to the order of traversal. This will still give k iterations, the same as without the auto-expanding bounding box, i.e., it does not get worse with the auto-expanding bounding box than without it.

The average scenario is difficult to predict precisely since it depends largely on the data. If the point cloud is dense, there is a high chance of adding new points early

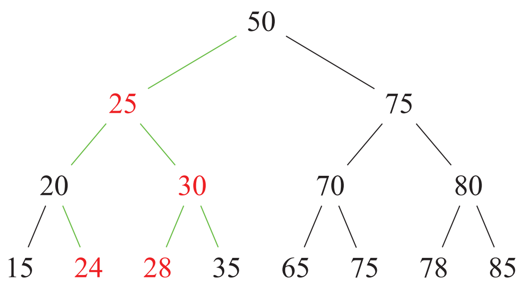


Figure 3.13. Marking subtrees as removed. Red – found nodes, green – search path

in the search, and expansions are more likely. If the point cloud is very sparse, for example, there are fewer points found during a single iteration than the required amount of bounding box expansions, the improvement will yield a diminishing performance increase.

Figure 3.13 visualizes an example of a good scenario of the expanding bounding box. The initial bounding box is $[25 - 30]$, and the bounding box expansion sensitivity equals $h = 2$. Then the following nodes are analyzed:

1. Node '25' is found; the search range is updated to $[23 - 30]$.
2. Node '24' is found; the search range is updated to $[22 - 30]$.
3. Node '30' is found; the search range is updated to $[22 - 32]$.
4. Node '28' is found; the search range remains unchanged.

A second iteration will be required, but it will find the same points again and conclude that the search is finished.

The nature of node traversal in point clouds has some impact on the bounding box expansions. It has already been noted that the algorithm performs best if the bounding box is expanded as much as possible during a single iteration. The search tree traversal is usually performed from the left to the right. However, other options are also available. For example, the root may be checked first, and the child nodes can be checked after that (a combination of left-to-right search and top-to-bottom search). A full top-to-bottom search is also available, however, it is not recursion-friendly. Since the search tree is a recursive data structure, it is efficient to use recursive algorithms for it. If the search is performed fully left to right, the first captured node will be the left-most node available for the bounding box. Such a value has a high chance to be closer to the boundary of the bounding box because the extremes of the search tree tend to appear near the sides of the tree. However, this is close to the extreme of the initial bounding box, so it might miss the values lower than the initial bounding box boundary completely. On the other hand, selecting the root node before the children does not suffer from this issue as much because the left-most values will be analyzed later in the search, usually after $\log n$ nodes (the path is root – left child – left sub-child, etc.). However, this has to be confirmed experimentally since real large search trees are complex and difficult to analyze theoretically.

Combination of improvements

It has already been mentioned that marking tree nodes as removed and marking whole branches as fully removed are related as the latter cannot be implemented without the former. Tree node removal can be implemented separately. The automatic bounding box expansion is orthogonal to the other improvements in the sense that it can be applied in combination with the others as an extra measure or separately. This is possible because the improvements exploit different properties of the search tree and the search itself. The node and branch removal techniques are possible because the tree allows an extra state per node at a cost of extra memory to hold it and the fact that the same node cannot be added to the results twice. The auto-expanding bounding box, on the other hand, depends on the mutability of the bounding box, which is an external state to the search tree, and the property of the minima defined in Formula (3.16). Since these properties do not depend on each other, it is possible to combine the improvement techniques.

Combining the auto-expanding bounding box with the marking nodes as removed combines the benefits of both improvements. This also guarantees the same bounding box expansion path as visiting the same node for the second time will not change the bounding box which has already been expanded according to this point. The same applies to marking whole branches as removed – if a branch has been removed, it means that the bounding box has already been expanded according to the points of that branch. On the other hand, the size of the bounding box does not change the fact that if a point has been found, there is no need to visit it again. Therefore, all techniques can be combined without one negatively impacting the other in any way.

The full final algorithm is outlined as Algorithms 1 - 3. Its parameter h is the hyperparameter which defines the bounding box increment size.

Algorithm 1 Algorithm to find a cluster

```
Require: $h > 0$ // bounding box increment size
Require: firstPoint $\in$ pointCloud
Output: cluster
 1: currentPoints = [firstPoint]
 2: boundingBox = point.coordinates $\pm$ h
 3: newPointsAdded = true
 4: while newPointsAdded do
 5:     closePoints = fcp(boundingBox,h,root)
 6:     if closePoints.size = 0 then
 7:         newPointsAdded = false
 8:     end if
 9:     currentPoints $\cup$= closePoints
10: end while
11: return cluster(currentPoints)
```

Algorithm 2 Algorithm to find close points

```
Output: points
 1: procedure fcp(boundingBox,h,node,result=[])
 2:     if node.depth = 0 then
 3:         currentLocation = node.location.x
 4:     else
 5:         if node.depth = 1 then
 6:             currentLocation = node.location.y
 7:         else
 8:             currentLocation = node.location.z
 9:         end if
10:     end if
11:     if !node.removed and boundingBox.contains(node) then
12:         result ∪= node
13:         node.removed = true
14:         boundingBox = expand(boundingBox, node, h)
15:     end if
16:     result ∪= fcp(boundingBox,h,node.left,result)
17:     result ∪= fcp(boundingBox,h,node.right,result)
18:     return result
19: end procedure
```

Algorithm 3 Algorithm to expand the bounding box

```
Require: $h > 0$ // bounding box increment size
 1: procedure expand(boundingBox, node, h)
 2:     boundingBox.minX = min(boundingBox.minX, node.x - h)
 3:     boundingBox.maxX = min(boundingBox.maxX, node.x + h)
 4:     boundingBox.minY = min(boundingBox.minY, node.y - h)
 5:     boundingBox.maxY = min(boundingBox.maxY, node.y + h)
 6:     boundingBox.minZ = min(boundingBox.minZ, node.z - h)
 7:     boundingBox.maxZ = min(boundingBox.maxZ, node.z + h)
 8: end procedure
```

3.4.2. Full-manual point cloud segmentation

Segmentation can be performed by humans fully manually. This is the most straightforward solution to the segmentation problem. Its advantage is accuracy – a human can easily identify a distinct object. This is also evident when looking at Figure 4.1. It does not take any sizable effort for an average person to identify that each image contains a human body and the location of the body. The boundaries are also quite clearly visible in *Kinect* depth images preprocessed by a bilateral filter.

The only challenge is to efficiently get the object segmentation mask from the person who performs the segmentation. It is clear from the current state of the art that this process can be extremely time-consuming for a human. This is only one of the reasons why there is a limited amount of datasets for segmentation – they are very time-consuming to create. On the other hand, automatic segmentation may produce incorrect output. This also requires human input. In the case of depth images with everyday objects that are distinct in space, depth maps were selected as the data representation for manual segmentation. It is easy to process for a human and to visualize for a computer (it is a regular image on the screen). Segmentation errors are also visible and can be identified easily. However, a tool to mark segments manually is required to complete this task.

One of the classic ways to mark something on the screen is by using a mouse cursor. Selecting an area of the image with a mouse while holding its left button down is a common and intuitive way of selecting a segment. However, depending on the situation, differently sized areas must be marked. If the area is large, a large 'brush' allows marking it quicker. If the area is small or the shape is complex, a smaller 'brush' would be more useful. Therefore, it was decided to implement three square-shaped brushes with sizes of 5x5, 10x10, and 15x15 pixels of the depth map. The user may choose their own best size depending on the situation. Internally, the point cloud search tree has to have the same state as if the segment marked by the user was marked by any automatic algorithm. This allows combining multiple techniques – if automatic segmentation over-segments the object, the user may add a cluster to the segmentation output manually. Alternatively, if the algorithm tends to under-segment an object, the user may mark an edge of the object, and this would work as a virtual boundary – the segmentation algorithm would skip the points marked by the user since they already belong to the object.

The internal state update involves finding all points selected by the user and changing their state to 'removed'. There are two ways to do this. The points can be marked one by one. If the user marked k points, since the search complexity in a tree of size n is $O(\log n)$, the total marking complexity would be $O(k \log n)$. Alternatively, it could be done by passing the whole point list and traversing the whole tree once. If the list is large, it should be sorted, which is $O(k \log k)$ operation, to utilize binary search, which is $O(\log k)$ operation. The search would be performed n times. This means that the complexity of this approach is $O(k \log k + n \log k)$. Since k is usually much smaller than n, the first component has a diminishing weight and can be skipped. The first approach is better than the second when

$$k \log n < n \log k \tag{3.17}$$

Since $k \leq n$, k can be expressed as $k = \gamma n, 0 \leq \gamma \leq 1$. Thus, the first approach is better when

$$\gamma n \log n < n \log(\gamma n) \tag{3.18}$$

or, if simplified

$$\gamma < \log \frac{\gamma}{n} + 1 \tag{3.19}$$

Since γ is at most 1 and n is at least 1, $\log(\frac{\gamma}{n})$ is at least 0. The whole right-hand expression is then at least 1, which means that the first approach is better always except when $k = n$ as they are equal in this case.

Another common operation is the removal of segments marked by mistake. This can be done the same way as marking segments. The user selects an area to exclude from segmentation by using a mouse, and the selection is removed from the segmentation output. However, this carries some extra information compared to just selecting a segment on the first try. Since the area is specifically selected as excluded, this means that it cannot belong to the final segmentation output. Therefore, if combined with the automatic segmentation methods, this zone should be excluded anyway. This extra information can be preserved to prevent mis-segmentation later. One of the ways to do it is to mark the 'erased' as removed but not include them in the segmentation output. This prevents analyzing the nodes later, and they are not added by the segmentation algorithm. However, if segmentation is again re-added manually, this overrides the erased segmentation.

The full-manual segmentation activity diagram is depicted in Figure 3.14. The user selects the brush size and type (*draw* or *erase*) and marks an area. This is repeated until the segmentation output is satisfactory. The system either adds or removes the selected points from the segmentation output, marks the points removed, and updates the segmentation output shown to the user.

3.4.2.1. Combining semi-automatic and full-manual segmentation

Automatic and manual segmentation techniques can be combined into a single segmentation workflow. Related research has shown that, while manual segmentation is the most accurate, automatic segmentation is much faster. A combination of manual and automatic segmentation has also been proven to be a successful technique. As a result, these techniques are also combined to implement a segmentation solution for binary human body segmentation.

The propositions made in the previous subsection lead to the following segmentation possibilities:

- Run automatic segmentation by selecting an initial point;
- Add a segment to the point cloud manually;
- Add an 'erased' segment to the point cloud manually;
- Undo an action.

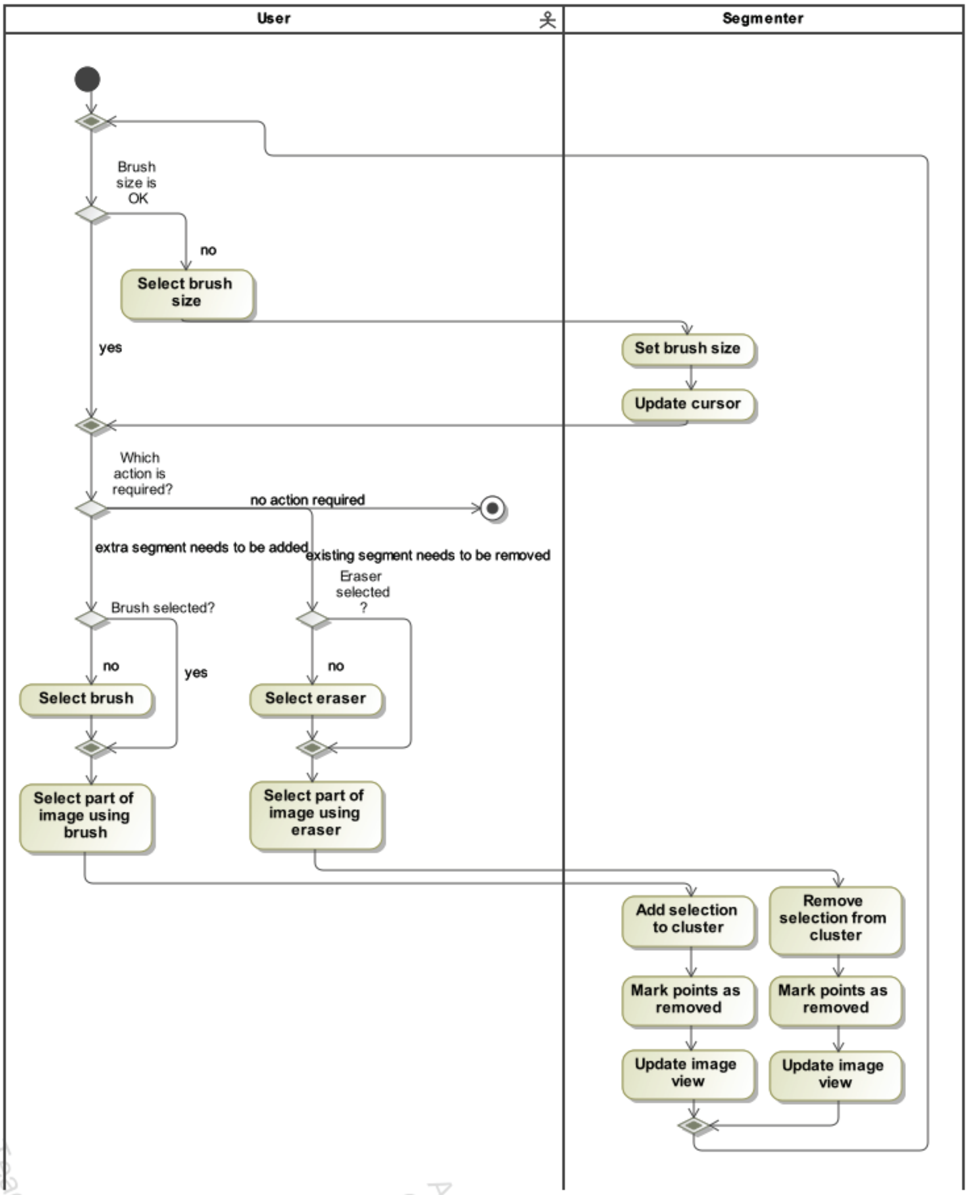


Figure 3.14. Fully manual segmentation UML activity diagram

These techniques can be combined to get the correct segmentation output as fast as possible. Automatic segmentation is faster than manual segmentation in all related research, therefore, it may be worth trying the automatic solution and then correcting its output. Automatic segmentation may also be applied several times. This is useful in the case of over-segmentation. Given a selected segment A and ground truth G, if $A \subset G$, or at least $G \setminus A \neq \emptyset$, there is a segment in G that can be added to the segmentation output by applying automatic segmentation on one of its points. This should be faster than trying to add the segment manually.

Manual segmentation is an alternative way to correct the output of the segmentation. If automatic segmentation fails to capture a part of the object, it can be marked manually. This may be useful for scenarios where automatic segmentation heavily under-segments a small part of the object – it may then be easier to mark it manually instead. Manually marking a segment may also be useful to create boundaries in space to help segment the rest of the object automatically. It has already been explained above that finding a segment in the point cloud marks the selected tree nodes as removed. The nodes are ignored during further segmentation and do not expand the bounding box. If the segment is selected manually, the same rules apply. Therefore, if the user suspects why under-segmentation has happened, they may draw a boundary segment and prevent this under-segmentation.

Erased segments work similarly to manually marked segments. The only difference is that their points are specifically excluded from the segmentation output. Their primary use is to correct under-segmentation. If a part of the segmentation output does not belong to the object, the user may manually mark this segment as erased, and the segmentation will be excluded from the output.

3.4.2.2. Extending the segmentation to video sequences

If the frames originate from the same depth video sequence, they are most likely related to each other. This is an important property for analyzing video sequences. Let us say that a video sequence S consists of frames $s_1, s_2, \ldots, s_n$, where the indices correspond to the order number in time, i.e., a higher frame number means that it was taken later. The *Kinect 2* sensor has a fixed frame rate of 30 frames per second. Therefore, each frame is taken 33 ms later than the previous frame. This period is short in macroscopic measures. The normal human walking speed at ages 20 to 69 years is about 1.3 m/s, as shown via research conducted by Schimpl et al. [332]. This means that a human is expected to walk about 4.3 cm between two frames. This is a small change compared to the size of the human body. If the video sequence is static, the changes will be even smaller. Therefore, frames s_i and s_{i+1} can be considered similar to some extent. It could then be hypothesized that if two frames are similar, the same segmentation inputs should produce similar segmentation outputs.

This hypothesis leads to a solution for segmenting video sequences quicker. It has already been described above that image segmentation is the total output of automatic and manual segmentation steps. Let us say that the segmentation process P consisted of actions $p_1, p_2, \ldots, p_n$. Each action p_i is either automatic segmentation with user-provided parameters h and m, manual segmentation, or manual erasing. The full output of the segmentation is the result of a pipeline of actions with a depth image as the input. These actions can be repeated for the next frame in the same order to produce a similar result. This does not guarantee the correct segmentation, however, the chance of it being correct should correlate with the similarity of the frames. Even further, if the scene is static, the same segmentation pipeline can be applied to multiple images. If the whole sequence is static, the same segmentation pipeline may even work for the whole sequence.

An alternative strategy would be to use the whole bounding box from a previous frame, however, when manual annotations are involved, the resulting shape cannot be described by using a bounding box. If two objects touch each other, their bounding boxes are likely to overlap, however, the objects are distinct. Simply drawing a bounding box in the next frame would ignore this fact and result in the same mistakes in each frame.

With these assumptions and limitations in mind, the solution shown in Figure 3.15 was implemented. The user can see multiple images on the screen at once. When the first frame is segmented via the selected techniques, the user selects to transfer segmentation to the subsequent frames. The segmentation is repeated for all the subsequent frames that are visible on the screen. This is done by applying the same segmentation pipeline for each frame. Segmentation is repeated by treating the depth frame as an image. Automatic segmentation is transferred by selecting the same point m in the 2D image space. Manual segmentation is transferred by selecting the points that correspond to the same pixels in the 2D image space. The output may then be updated further by the user if needed, but this should give a viable starting point for faster segmentation.

Since the video sequence consists of more frames than is visible at once, the segmentation view is paginated. When the user is done with the current page, the segmentation is also automatically transferred to the whole next page. The last frame is selected as the base segmentation pipeline provider since it should be the most similar to the frames on the next page. If the segmentation is OK for the whole page, the only required thing is to validate the output and go to the next page. In the ideal scenario, the user would select automatic segmentation, click on the object, transfer segmentation to all frames, cycle through the pages, and the segmentation would be finished for the frame sequence.

3.4.2.3. Working with large image sequences

The data structures described in Section 4.2.1 consume a noticeable amount of memory. The loaded frame is represented by a search tree and a depth map simultaneously. This totals at about 11.3 MB of memory per frame. In addition to that, the segmentation output also has to be stored in the memory. The size depends on the segmentation output – a list of references to points is stored where each reference takes 8 bytes of memory. In the case of a segment that is 10% of the original point cloud, this is an extra 170 kB of memory. The whole application, however, requires more memory to show the depth visually. Without this extra overhead, a 300-frame-long video sequence (10 seconds at 30 frames per second) would take ~3.4 GB of mem-

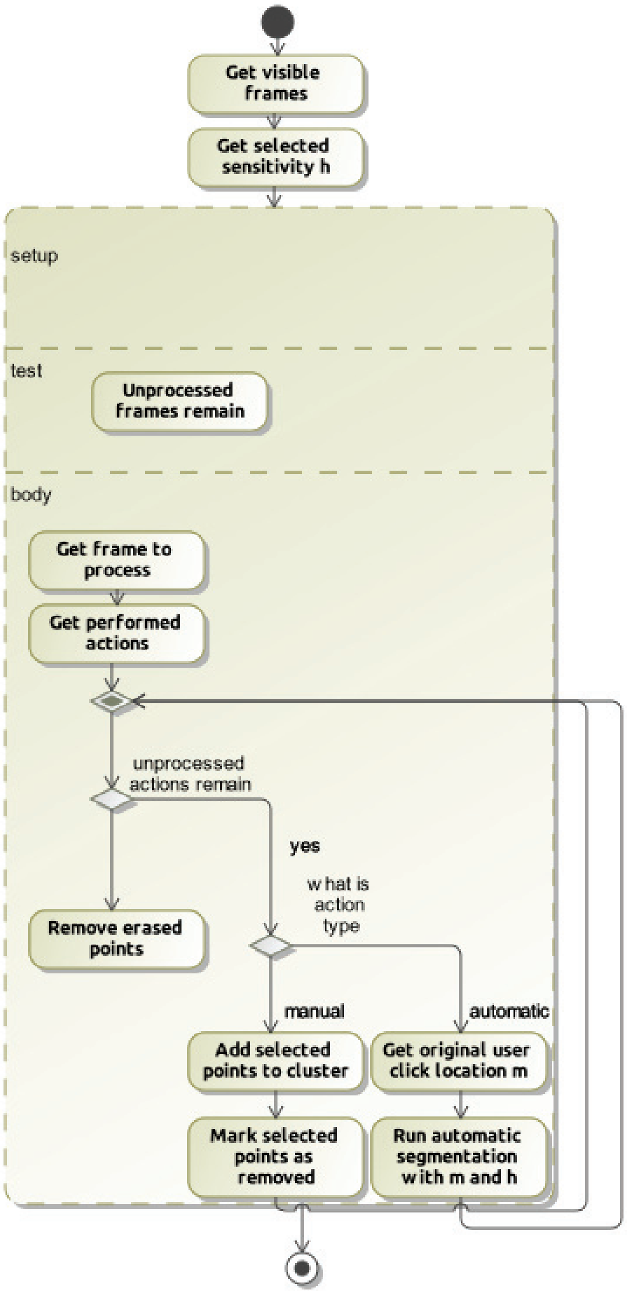

Get visible frames
Get selected sensitivity h
setup
test
Unprocessed frames remain
body
Get frame to process
Get performed actions
unprocessed actions remain
yes
Remove erased points
what is action type
manual
automatic
Add selected points to cluster
Get original user click location m
Mark selected points as removed
Run automatic segmentation with m and h

Figure 3.15. Mass segmentation UML activity diagram

ory. Since the software is implemented in Java, this memory requirement should be somewhat relaxed to prevent frequent garbage collection. Related research showed that a performance penalty in memory-hungry applications, where large datasets are processed, garbage collection may contribute to low performance [333]. It was experimentally found that an increase of the memory to ~42 MB[3] per frame greatly reduces the runtime of the segmentation tool (this includes all other parts of the system, not just the segmentation classes). Therefore, 12.6 GB RAM is needed to segment a full 300-frame sequence without performance degradation.

This memory size requires working with the point cloud in batches. The batch size is determined by checking the memory limit of the application (provided by `-Xmx` JVM parameter) and dividing it by 42 MB. The user may change this parameter if required. The frames are then loaded to the memory until the limit has been reached, processed by the user, and the output is saved to the disk in a temporary files folder. When all the batches have been processed, all segmentation outputs are loaded (they are pairs of the original depth and the binary mask, 340 kB per frame), concatenated into a single sequence, and recorded in the disk as one sequence of frames.

3.4.3. Under-segmentation reduction using A recursive 2-Means split algorithm with a random forest classifier for split acceptance

This section provides a solution to the under-segmentation problem by cutting a part of the segmentation output. This is achieved by a proposed modification of the classic K-Means algorithm:

- The clustering is repeated recursively with two clusters instead of testing the output with a different amount of clusters;
- The best amount of splits is decided by a random forest classifier from the proposed metrics.

The following sections provide the details of all the employed algorithms.

3.4.3.1. Under-segmentation problem

Section 4.2.5.1 provides the experimental findings of applying the Euclidean-search-based algorithms for human body segmentation. These results showed that the algorithms are prone to under-segmentation (see Figure 4.10). Formally, the output seen in the examples can be defined as

$$A_T \cap A_B = \emptyset : A_s = A_T \cup A_B \tag{3.20}$$

Here, A_T is the actual human body (the ground truth, which is also a subset of the segmentation output), A_B is the background (the false positive part of the segmentation output). Then there exists such function R that

$$A_T = R(A_s) \tag{3.21}$$

[3]This memory constraint may or may not be related to the fact that 42 is the answer to life, the universe and everything [334, 335]

The goal is to find an approximation of the function R that would cut the false-positives from the output of the segmentation in the case of under-segmentation. The algorithm that approximates the output of function R is presented in the following sections.

3.4.3.2. 2-Means split and metrics of the split

One possible way to find function R is to apply K-Means clustering and reject clusters that are not human. However, parameter k is unknown beforehand since the number of the captured objects is variable. Another challenge is to determine whether the clustering output should be rejected at all – the clustering may provide worse results than without it, or the split may not even be required at all if the segmentation results are already good.

The first problem can be solved by a recursive 2-Means split algorithm. In the area of application (semi-automatic human body segmentation), one point is already known to be a part of the human body. This point will act as a fixed, non-moving cluster center. Since the number of the required clusters k is unknown, a modification of the Elbow algorithm [237] is proposed. The segmentation output A_s is split into two clusters by using K-Means (hence the name 2-Means) recursively with the 'correct' part of the cluster A_T. The initial point for the second centroid is selected as the point that is the furthest away from the first cluster center. A regular K-Means with 2 clusters is performed with the exception that only the second centroid is updated. The algorithm halts when it has converged (the second centroid does not move) or after 10 iterations to prevent a long runtime. The elbow algorithm would then add another cluster and look for convergence of the cost function. Instead, this dissertation proposes a different solution – we estimate whether the split has improved the segmentation accuracy or not, and if yes, repeat the same 2-Means clustering with the cluster that contains the initial point. Otherwise, the clustering halts.

The remaining challenge is to estimate the quality of the split. There are five pieces of data to analyze:

- The full cluster of points (the input for the split).
- The sub-cluster that includes the initial point.
- The sub-cluster that does not include the initial point.
- The original user-selected point (the first centroid).
- The second centroid.

There is no more data to analyze, so the information about the quality of the split must exist within this data. Intuitively, the cluster sizes and their positions might contain the answers to this problem. The distance between the clusters and the proportions of the points could be used to infer the quality of the split. For example, it is evident from the segmentation error analysis that, in the worst cases of under-segmentation, the background segment would be a lot larger than the human itself. The human body also has a defined body shape which may impact the distances between the first centroid and the cluster points since only the human body surface points would be present in the point cloud. Therefore, 8 pieces of data are extracted for further analysis:

- The average distance between all combinations of two centroids and three clusters (six distances).

Table 3.3. Random forest classification report

	Precision	Recall	F1-score
Incorrect cuts	0.94	0.90	0.92
Correct cuts	0.95	0.97	0.96
Accuracy		**0.95**	

- The sizes of both clusters (two sizes).

Their values were selected because they seem to represent different types of mis-segmented clusters. If the original segmentation captured a single object, the distances between both point clouds and both centroids will be much lower than in the case of both clusters representing two distinct objects. If the whole background is captured, the false cluster will also be much larger than the true cluster. If two very distinct objects are in the cluster, the average distances between the sub-clusters and their centroids will be much smaller than the the average distance between either sub-cluster and the other centroid. These metrics are also relatively fast to compute. Unfortunately, the rules for split correctness are difficult to analyze.

3.4.3.3. Random forest classifier for split acceptance

Since the rules to measure the quality of the split are difficult to determine, they could instead be learned by a machine learning solution. This is a classification task where the inputs are the metrics described in the previous subsection, and the output is a decision on whether the split described by those features improves the accuracy of the segmentation or not.

Training data has been collected by using the dataset of depth data and the ground truth masks described in Section 4.1. The bounding-box-based segmentation was run, and 2-Means splits were applied until the accuracy after the split became lower than before the split. Splits were limited to a maximum of 10 per frame. All intermediate samples were added to the training data collection. The training data collection consists of 8 metric values and a Boolean label indicating if the split should be accepted or not. The data acquisition activity diagram is shown in Figure 3.16. This is repeated for every frame.

This methodology allowed fully-automatic training data acquisition after the labeled data became available. However, data proportions by classes depend on the data and the segmentation quality. 72% of the data was for correct splits, whereas 28% for incorrect splits. The total amount of samples used for the training was 70.4k.

The data was split into the train (80%) and test (20%) sets video-wise. This type of split was selected to prevent too similar data in the test and train datasets since two frames in the same video sequence may be almost identical. 9 tree estimators were used to train the random forest because a higher amount of tree estimators showed no improvement in accuracy, while a lower number of estimators even degraded the accuracy. The classification report after the training is shown in Table 3.3.

The classifier achieved 95% accuracy. Correct cuts are classified better by all

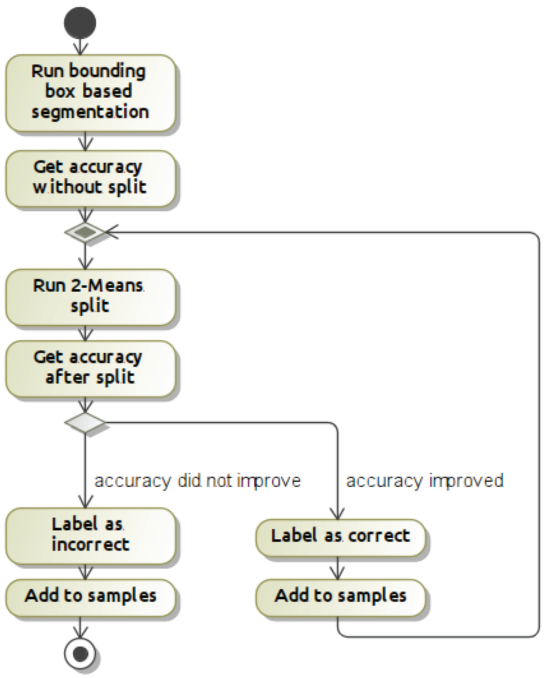
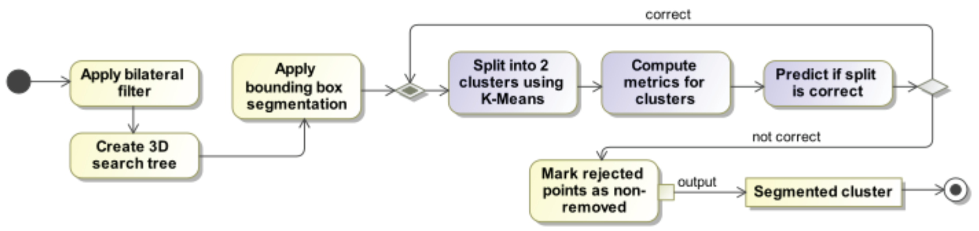


Figure 3.16. Training data acquisition activity diagram

metrics, and this may have to do with the imbalance in the training data – the correct cut sample size was larger. Despite that, incorrect cut detection has 90% recall, which is the lowest number in the table. This means that 90% of all incorrect cuts were labeled correctly, and other metrics are higher than this. 97% of all correct cuts were classified correctly, which means than only 3% of the correct cuts were rejected. The results suggest that this classifier is suitable for cluster split quality classification and the metrics were selected correctly.

3.4.3.4. Integrating the correction into semi-automatic segmentation workflow

Figure 3.17 shows how this proposed segmentation correction algorithm can be incorporated into the semi-automatic segmentation workflow. The purple blocks show the newly added steps. The segmentation is applied as usual, and the segmentation output is split into 2 clusters. The metrics are computed for the split, and the random forest classifier predicts whether the split would improve or reduce the accuracy. The process is repeated until the prediction comes as reducing the accuracy, and the last split is rejected. Since the segmentation step marked some points as removed, and now they have been rejected as not correct, the state of the search tree nodes has to be reset as well.



correct
Apply bilateral filter
Create 3D search tree
Apply bounding box segmentation
Split into 2 clusters using K-Means
Compute metrics for clusters
Predict if split is correct
not correct
Mark rejected points as non-removed
output
Segmented cluster

Figure 3.17. Segmentation correction using recursive 2-Means splits with a random forest classifier integration in semi-automatic segmentation

3.4.3.5. Performance implications

Since the points have to be marked as non-removed, this adds some extra work. It was shown in Section 3.4.2 that the algorithmic complexity of this procedure is $O(k \log n)$, where k is the number of points to mark as non-removed, and n is the total amount of points in the tree.

Further extra work to do is the computation of the 8 metrics required to make predictions, and splitting the clusters into two sub-clusters. Six distances have to be evaluated. They are all computed by using not only the same principle, but even the same Java method with two parameters – the point (centroid) and the cluster. It computes the distance for each point in the cluster. The size of the original cluster is m, and the sizes of sub-clusters m_c and m_s also total m. Therefore, the total amount of work for the distance metrics is $O(4m)$. The cluster sizes are computed to have $O(1)$ algorithmic complexity. Therefore, extra work due to the metric evaluations has $O(4m)$ complexity.

Another piece of extra work is the 2-Means split itself. It involves the following steps:

1. Get the furthest point from the initial point.
2. Partition points by centroids 1-10 times.
3. Get a new centroid in each partition.
4. If the split improves the accuracy, a collection of all excluded points is updated.

Finding the furthest point involves checking the distance for each point in the initial cluster, hence complexity $O(m)$. Partitioning points by the centroid also involves one iteration over the initial cluster and adding the point to one of the two lists (the first or second cluster), hence complexity $O(m)$ again. The new centroid only has to be evaluated for one cluster and involves checking all the points in that cluster, which has complexity $O(m_s)$. Updating the list of excluded points is also $O(m_s)$ operation. The three final operations can be repeated up to 10 times, however, this still leaves the total complexity at $O(m + m_s)$.

There is also some extra time required to make the prediction, however, the prediction time is constant. The prediction is made by using the *JPMML-Evaluator*[4] library for the Java programming language: the model is loaded from a PMML file

[4]`https://github.com/jpmml/jpmml-evaluator`

(during the startup once), and then predictions are made. The model definition takes 6.09 MB of the disk space.

3.5. Fully-automatic segmentation

Agrast-6 (Abridged VGG-based Reflected Architecture for Segmentation Training with 6 encoder layers) is proposed to solve the fully-automatic segmentation task.

3.5.1. Theoretical basis and hypothesis

The *SegNet* neural network [50] is one of the most successful neural networks for image segmentation. It consists of an encoder and a decoder where the encoder is based on the *VGG-16* network and the decoder is the reverse of the encoder. Since *VGG-16* itself is a deep neural network, adding a decoder makes *SegNet* even deeper. The network has 26 layers which allow good accuracy, however, the architecture itself is huge and slow. It does not use any fully-connected layers, but the model size is still 117 MB, and a forward pass was estimated to take 422.5 ms with 360×480 resolution images and NVidia Titan GPU.

While *SegNet's* segmentation accuracy for RGB images is good, it solves multi-class segmentation for RGB images. The model originally was trained to recognize 37 classes of objects. This means that the features for those 37 classes have to be stored inside the model. On the other hand, the binary segmentation task only requires one class to be learned. This research focuses on the human body binary segmentation, therefore, it has to only learn the features of a human. Moreover, the data that the *SegNet* neural network processes is RGB, which means that the input consists of three channels, and the features have to be inferred from them all. The depth data, on the other hand, is simpler with only one channel that carries all input information.

It is possible to use the *SegNet* neural network for binary human body segmentation. The number of classes has to be reduced to 2 (human and background), and the depth channel is repeated across all input channels. However, it could be hypothesized that a smaller, simpler architecture of the same type is possible. Smaller architectures usually mean a smaller model size and a shorter inference time, therefore, such an architecture could run faster or on lower-spec devices. The results could be achieved by learning more aggressively as *SegNet* uses 13 layers for the encoder, but fewer features could be learned when using a shorter network while keeping the other properties of the network.

3.5.2. Proposed neural network – Agrast-6 architecture

The proposed neural network architecture is an Abridged *VGG-based* Reflected Architecture for Segmentation Training (abbreviated as *Agrast-6*[5]). *SegNet* neural network is used as the ideological basis. *SegNet* itself is an encoder-decoder architecture based on *VGG*, therefore, Agrast-6 is also based on *VGG*. It inherits the encoder-decoder architecture where the decoder is a reflected (mirrored) encoder. However, Agrast-6 is designed to be more lightweight, therefore, it is an abridged version com-

[5]'Agrastas' means 'gooseberry' in Lithuanian, this abbreviation may or may not be intentional.

pared to *SegNet*.

VGG-16 consists of 5 blocks of layers. Each block has 2 or 3 convolutional layers with a 3×3 filter. The blocks also have a max-pooling layer at the end and reduce the dimensions of the image by a factor of 2. The depth dimension of the convolutional layers increases from 32 at the first layer and ends with 512 at the last layer. This high dimensionality allows the network to learn more features. However, it is hypothesized that there are fewer features to learn for binary segmentation. Therefore, the *VGG* architecture is abridged in Agrast-6. The following simplifications are proposed:

- Three blocks of layers instead of five.
- One convolutional layer per block instead of three or two.
- A lower depth of convolutional layers (32, 128, and 256 instead of up to 512).
- More aggressive max-pooling (dimensionality reduction of factors 4, 4, and 2).

The block reduction comes from the idea that there are fewer features to encode. The human body has distinct visual properties that are easily recognizable for a human as opposed to analyzing road segments, which was done by the *SegNet* neural network. Fewer convolutional layers come at a cost of fewer global features, however, the task being solved is to localize a single object in a depth image, which should not require as much global information as full multi-class segmentation. One convolutional layer per block also simplifies the network due to the same reasons. This reduction in the number of convolutional layers greatly reduces the complexity of the whole network. It should reduce the accuracy of the network, however, the author believes that the effect will not be large, and the model size reduction will be worth it. The convolutional layer depth reduction comes from the fewer features required to learn. The depth is required for *SegNet* to be able to learn more features, and this is simplified in Agrast-6. Finally, *VGG* and, consequently, *SegNet*, reduce the dimension of the original input by using max-pooling layers. Each layer reduces the dimension by a factor of 2, thus totaling the reduction of 32 times over all 5 blocks. This property is kept in Agrast-6 via more aggressive max-pooling. Since there are only three max-pooling layers, they reduce the dimensions by factors of 4, 4, and 2 to have the same total dimensionality reduction of 32. There are 6 layers in the encoder (3 convolutional and 3 max-pooling, not counting the input layer), hence the ‘6’ in Agrast-6.

The described first part acts as an encoder. The decoder is generated from the encoder automatically to ensure that Agrast-6 is a reflected architecture. The process is shown in the UML activity diagram in Figure 3.18. First, the encoder is constructed with the layers described in the previous paragraph. Next, the decoder is generated. For each layer in the encoder, a matching decoder layer is added in the reverse order. An upsampling layer is added for the max-pooling layer, a transposed convolutional layer is added for the regular convolutional layer, and a convolutional layer is added for the input layer. The upsampling and transposed convolutional layers are reflected operations from the encoder.

The final convolution is added to reduce the depth of the output back to one. In the end, a bounded ReLU layer is added with a maximum value of 1. This is a deviation from the *SegNet* neural network which uses a softmax activation layer at the

Table 3.4. Agrast-6 training software and hardware details

CPU	AMD Ryzen R9-3900X
GPU	NVidia GTX 1660 Super (6 GB VRAM)
Graphics toolkit	CUDA 10.1
Neural network accelerator	cuDNN 7.6.5.32
Neural network library	TensorFlow 2.2.0
Programming language	Python 3.8.10

end. Softmax tends to produce better results with multi-class classification tasks [265]. However, Agarwal et al. showed that ReLU performs better for binary segmentation problems [336]. Therefore, a ReLU layer is used for the final prediction. Every convolutional layer in the network uses padding to keep the dimensions of the layer the same size as the previous layer. Padding is done by using zero values on all sides of the image. The Agrast-6 architecture is visualized in Figure 3.19. The resulting model has 1.2M trainable parameters.

It is important to note that the *SegNet* neural network is characterized by its U-net-like structure which includes direct connections between the layers of the encoding and decoding parts of the same resolution. This helps to improve the accuracy and resolution of the segmentation results and has contributed to the success of *SegNet* in a variety of image segmentation tasks. In developing the Agrast-6 architecture, the decision to simplify the architecture by removing these direct connections was made. This made it possible to reduce the overall complexity of the network. While this simplification does not provide a possibility for the decoder to utilize features not captured by the decoder, the expected accuracy reduction should not be critical. Since the architecture is different, pre-trained VGG encoder weights are not utilized, either.

3.5.3. Agrast-6 training and training data split

Agrast-6 was trained by using the datasets described in Section 4.1. The ground truth labels were marked manually by two trained individuals and used for training. The input for the model is the original depth image, and the expected output is a binary mask where a human silhouette is marked as the foreground and other objects are marked as the background. The *Kinect* depth frame dimensions are 424×512, however, Agrast-6 downscales and then upscales an image by a factor of 32. 424 does not divide by 32, so the data was right-padded with zeros for depth images as well as ground truth masks.

The model was implemented by using the TensorFlow 2 library for Python. It utilized GPU acceleration via the CUDA 10.1 library and the cuDNN 7 neural network acceleration library. The training process was performed by using NVidia GTX 1660 Super GPU. The software and hardware details are presented in Table 3.4.

The whole dataset was split into the training and testing parts. 80% of the data was used for training and 20% served for testing. The validation dataset was not used because the model was observed to consistently improve on the test set, and hyperparameters were tuned manually by observing the learning curves. K-fold cross-validation was not a viable solution, either, since the dataset is very large, and the

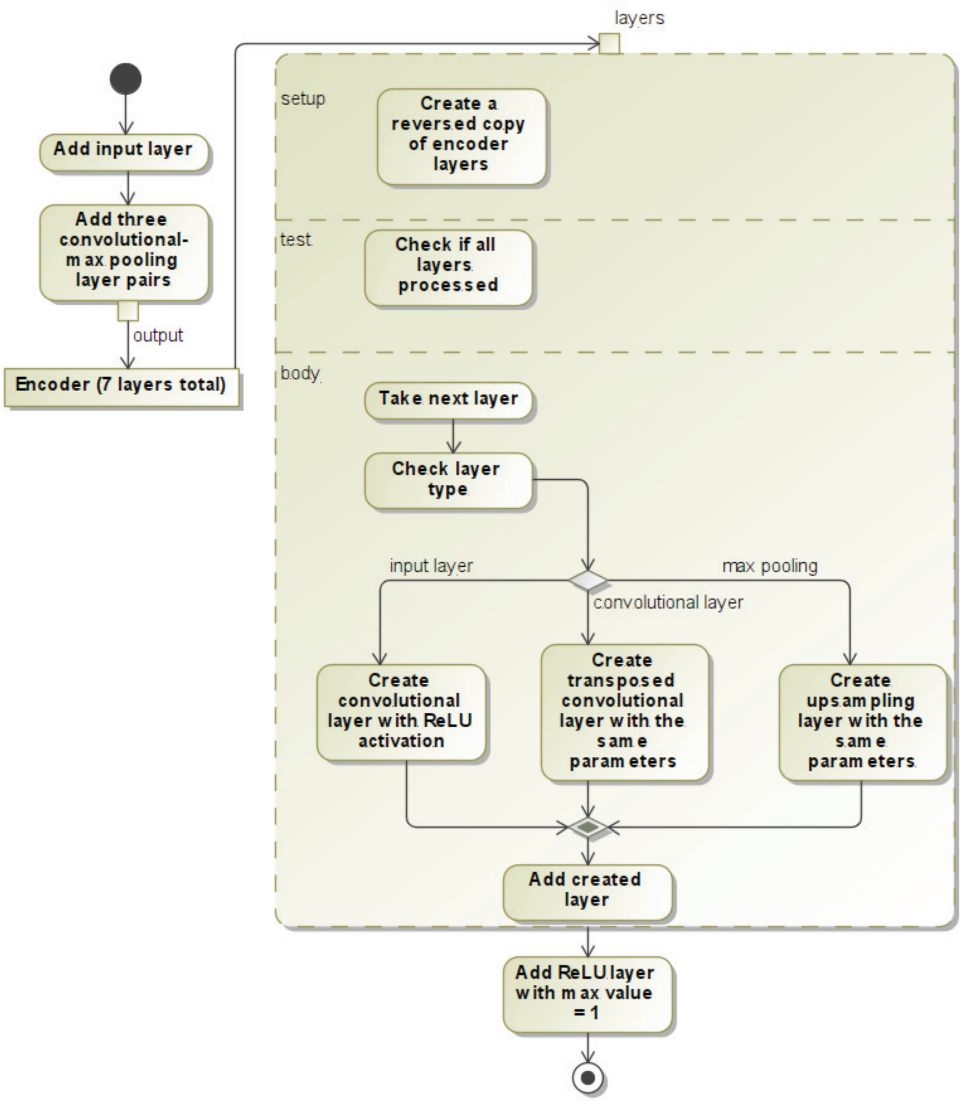

layers
setup
Create a reversed copy of encoder layers
test
Check if all layers processed
body
Take next layer
Check layer type
input layer
max pooling
convolutional layer
Create convolutional layer with ReLU activation
Create transposed convolutional layer with the same parameters
Create upsampling layer with the same parameters
Add created layer
Add ReLU layer with max value = 1
Add input layer
Add three convolutional-max pooling layer pairs
output
Encoder (7 layers total)

Figure 3.18. UML activity diagram of neural network creation process

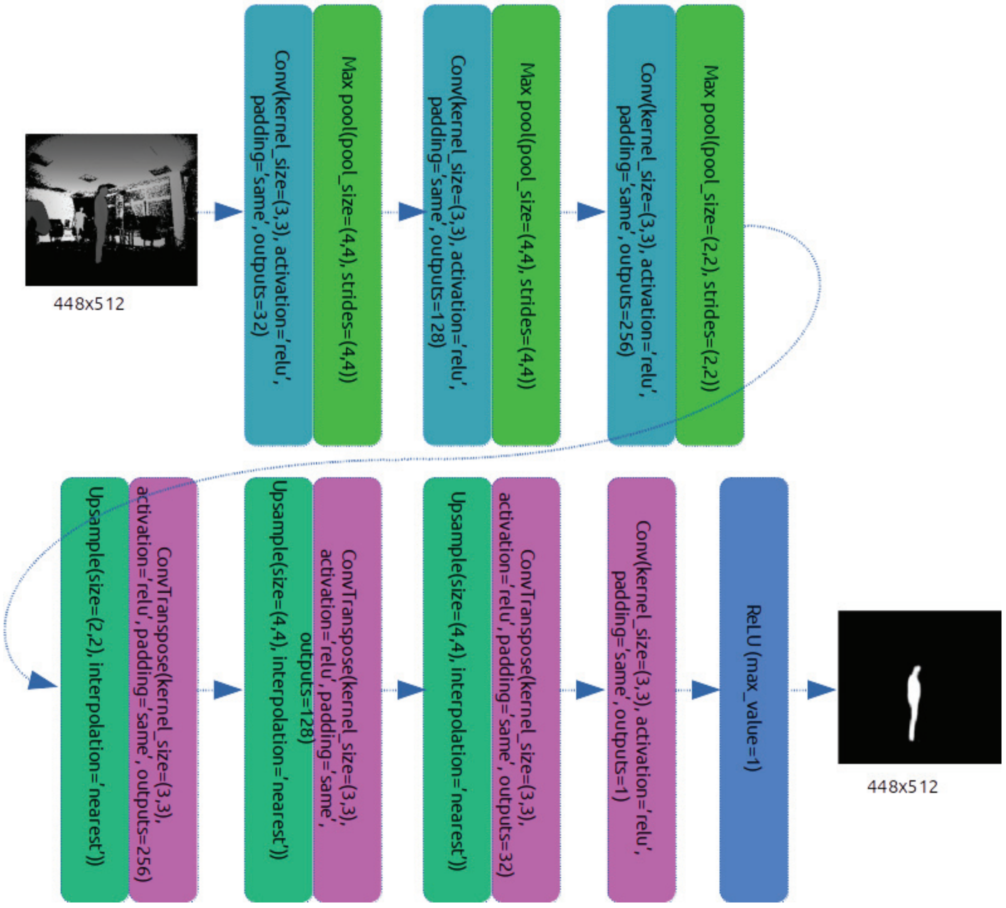

448x512
Conv(kernel_size=(3,3), activation='relu', padding='same', outputs=32)
Max pool(pool_size=(4,4), strides=(4,4))
Conv(kernel_size=(3,3), activation='relu', padding='same', outputs=128)
Max pool(pool_size=(4,4), strides=(4,4))
Conv(kernel_size=(3,3), activation='relu', padding='same', outputs=256)
Max pool(pool_size=(2,2), strides=(2,2))
Upsample(size=(2,2), interpolation='nearest')
ConvTranspose(kernel_size=(3,3), activation='relu', padding='same', outputs=256)
Upsample(size=(4,4), interpolation='nearest')
ConvTranspose(kernel_size=(3,3), activation='relu', padding='same', outputs=128)
Upsample(size=(4,4), interpolation='nearest')
ConvTranspose(kernel_size=(3,3), activation='relu', padding='same', outputs=32)
Conv(kernel_size=(3,3), activation='relu', padding='same', outputs=1)
ReLU (max_value=1)
448x512

Figure 3.19. Architecture of the proposed Agrast-6 model

Table 3.5. The values of Agrast-6 model hyperparameters

Hyperparameter	Value
Convolutional layer kernel size	3×3
Convolutional layer activation function	ReLU
Max-pooling pool size	4×4, 2×2 for final layer
Optimizer	Adam
Optimizer learning rate	0.0001
Loss function	Binary cross-entropy

training time would increase even further if this technique was applied. On the other hand, the large amount of samples introduces a lot of variance in the data since the frames are unique, hence, increasing it further between different epochs should not improve the ability to generalize by a significant margin. The Adam optimizer was used with a learning rate of 10^{-4}. This optimizer has been shown by multiple researchers to converge quicker than the alternatives, especially for neural networks [337, 338, 339]. The learning rate was selected by running the training process for a subset of data and observing the training loss. The learning rate of 10^{-4} was the highest learning rate that does not lead to unstable training loss as higher rates lead to training loss oscillation as the network is learning. Lower rates also worked, however, the training loss function was changing slower, which leads to longer training times. Binary cross-entropy was used as a loss function. This function is suitable for binary classification problems and is one of the most popular choices for such tasks [340, 341]. An overview of all hyperparameters is presented in Table 3.5.

The dataset used for model training was large, therefore, it could not be loaded all at once. The dataset size on the disk is over 150 GB. Therefore, it had to be loaded and fed to the network in batches. The data is organized as one file per depth video sequence. Unfortunately, reading and feeding the data sequentially leads to overfitting very quickly because the frames are very similar if they are in the same video sequence. Even shuffling the frames does not solve this problem. This was evident during the training process as the loss function value quickly jumps down as a new video sequence starts. This issue was solved by loading 20 video sequences at once (the full dataset is 942 sequences), concatenating their frames, and then shuffling them. When all the frames are processed, the next 20 video sequences are loaded, and the process is repeated. After this fix, the problem of in-sequence was solved, and the dips in the loss function did not occur anymore. The frames were fed to the training engine in batches of 4 as the video memory was quite limited on the GPU. The same procedure was used for the test dataset, except that the file batch size was 4 instead of 20 to reduce the memory usage. One epoch of training took 5 hours + 1 hour for testing against the test dataset.

3.6. Segmentation evaluation methodology

There are two categories of evaluated segmentation algorithms:

- Semi-automatic segmentation algorithms;

• Fully-automatic segmentation neural networks.

There is a baseline algorithm selected for both categories. The semi-automatic segmentation algorithms proposed in this dissertation are modifications of the Euclidean clustering algorithm, therefore, it is selected as the baseline. All algorithms, including the baseline algorithm, are implemented (or, in the case of the standard Euclidean clustering, reimplemented) in the Java programming language. In addition to that, a benchmark is also repeated with the PCL library directly, by utilizing its `EuclideanClusterExtraction` class. This benchmark uses the same performance measuring methodology, but it is implemented in the C++ programming language to match the technology used by PCL. Therefore, a two-way comparison is made: the performance of the proposed algorithms (Java) is compared to the implementations of the baseline algorithm in the same technology (Java), which makes a fair comparison, and the original implementation of the baseline algorithm (C++) serves to verify the validity of the results. However, it should be noted that the Java code is expected to have a lower performance. Informal benchmarks show that Java is about 2 times slower than C++ with binary tree problems [342].

Fully-automatic segmentation evaluation uses the *SegNet* neural network as the baseline because the proposed *Agrast-6* architecture is a modification of this neural network. An open-source TensorFlow implementation of *SegNet* is used [343] because *Agrast-6* is also implemented by using TensorFlow.

3.6.1. Accuracy evaluation methodology

The datasets described in Section 4.1 contain various scenes, people, and angles. On the other hand, the dataset is large, and the depth images that are captured in the same sequence are similar since the scenes are static. The scenes in different sequences can be different because different people are captured, different camera angles are used, and the people are in different poses. Every factor may impact the accuracy of the algorithms. It was decided to use 10% of all data by selecting every tenth frame from each sequence. This reduces the benchmarking time while preserving most of the variance in the data.

The ground truth labels were acquired by using the full software utilizing algorithms described in this dissertation. Two trained people performed the semi-automatic segmentation and produced the labels for the whole dataset.

The output of the algorithms depends on the parameters m and h (the manually selected point and search radius / bounding box size). Both parameters have to be provided during the benchmark.

A manually selected point has been simulated for each frame. Since the ground truth frames are known, the only hard requirement is to select a point that belongs to the ground truth segmentation output. It was decided to sort the points in 2D space by x coordinate. This is relatively quick to compute, guaranteed to belong to the ground truth, and has a low chance to be near the edge of the object vertically. This point was used as the initial starting point for the benchmark, which corresponds to the user-provided, parameter m.

Parameter h is difficult to automatically estimate, and this research leaves the selection of this parameter out of its scope. It is provided and adjusted by the user as they see fit. It was observed during the ground truth label collection that h values for these datasets most often were in the range $[2; 6]$. Given these values, it was decided to run the segmentation with all integer values $h \in [1; 10]$ and select the output that best matches the ground truth. This technique simulates the scenario where the best value is selected by the user.

The segmentation quality was evaluated by using cross-set intersection metrics. Since multiple segmentations with different h values were performed, the best segmentation result was determined by using this value as well.

Segmentation with the real-world data does not have to happen when using just one user click. It is common to select multiple points in the case of over-segmentation. This is certainnly easy for a human to perform and is considered a part of the normal segmentation process. For example, if the user waist pixel is selected, and the segmentation output did not include an arm, it is quick and easy to click on the arm. It would take much more effort to manually remove pixels that were under-segmented. Therefore, the benchmark simulates this behavior.

The whole segmentation process is shown in a UML activity diagram in Figure 3.20. Since the original algorithm applies a bilateral filter, it is also applied before segmentation. A 3D binary search tree is created from the filtered depth image. Each h value is represented by a different segmenter in this process. For each segmenter, a hash set of the ground truth indices and an empty list of clusters are created. A middle point is taken from the remaining ground truth index set, and the segmentation is performed using this point and the segmenter's h value. All found points are removed from the remaining ground truth index set and the cluster is added to a list of clusters. This process is repeated until either 90% of the ground truth points have been included in the output, or the algorithm has converged. This type of evaluation reduces the impact of over-segmentation, which is easy to fix by a human and exposes under-segmentation, which is difficult to fix for a human.

3.6.2. Performance evaluation methodology

The benchmark performs full point cloud segmentation as it divides the point cloud into as many non-intersecting clusters as possible. Some parts are equal for all implemented algorithms and their variations, for example, building the search tree. These parts are omitted in the benchmark, and only segmentation and algorithm-specific calculations are considered. The benchmark software was implemented in the Java programming language. The software and hardware specifications are listed in Table 3.6.

The benchmark is repeated for a total of 1000 randomly selected images from both datasets. Four example images are shown in Figure 3.21. Two metrics were calculated during the benchmark – the runtime of the algorithm, and the binary search tree node traversal count. Time was measured by using the Java's `System.nanoTime()` method and taking the difference between the times. The traversal count was determined by adding an extra state to the tree which that tracks the tree node traversals. The time metric depends on the hardware and software that the benchmarks are per-

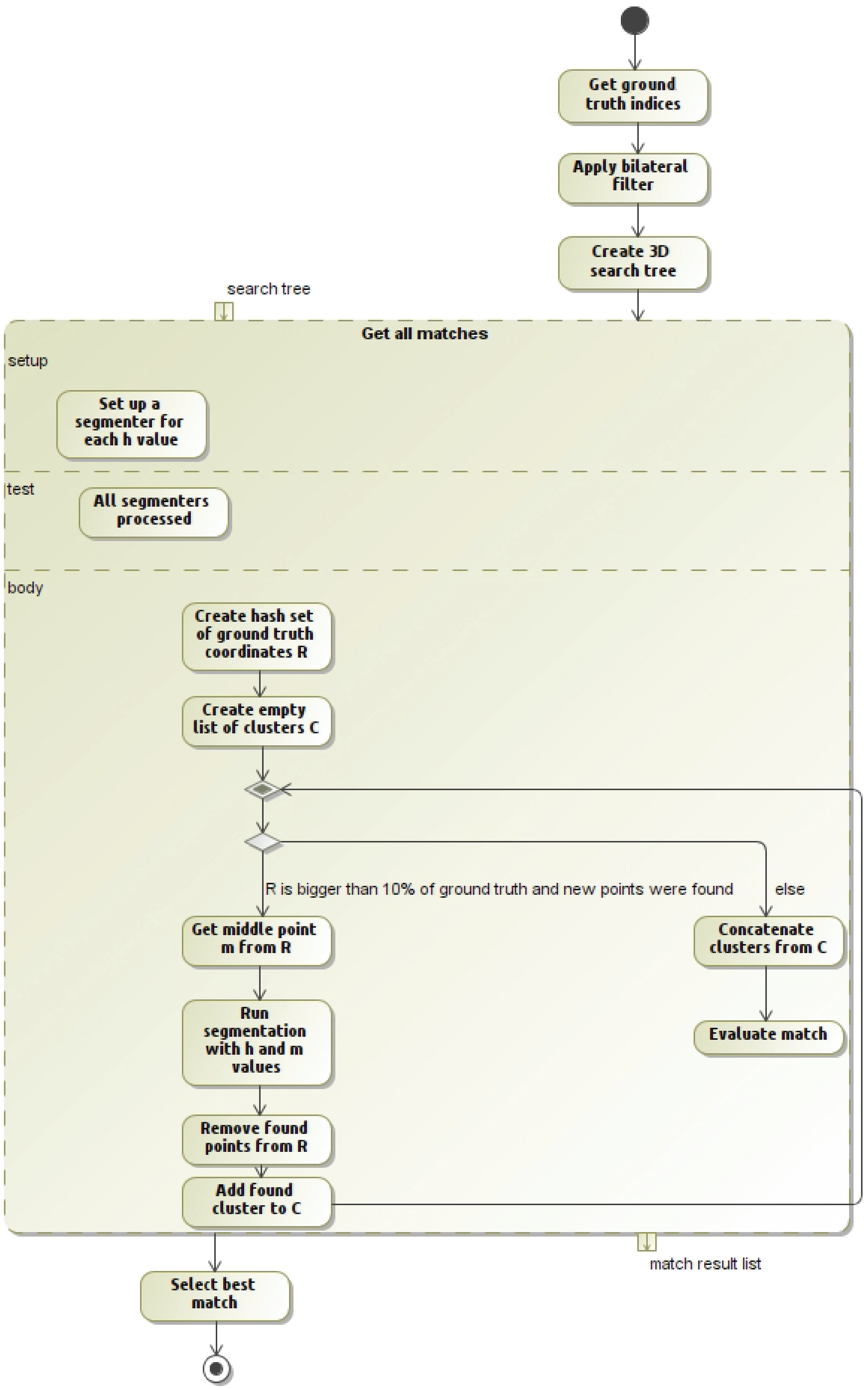

Get ground truth indices
Apply bilateral filter
Create 3D search tree
search tree
Get all matches
setup
Set up a segmenter for each h value
test
All segmenters processed
body
Create hash set of ground truth coordinates R
Create empty list of clusters C
R is bigger than 10% of ground truth and new points were found
else
Get middle point m from R
Run segmentation with h and m values
Remove found points from R
Add found cluster to C
Concatenate clusters from C
Evaluate match
match result list
Select best match

Figure 3.20. Accuracy benchmark UML activity diagram

Table 3.6. Benchmark software and hardware specifications

CPU	AMD Ryzen 9-3900X (820 GFLOPS)
RAM	2 x Kingston HyperX Fury Black 3200MHz 16 GB
GPU	NVidia GTX 1660 SUPER with 6GB of VRAM (5.0 TFLOPS)
Operating system	Ubuntu 20.04.4 LTS
JVM	OpenJDK 11.0.15

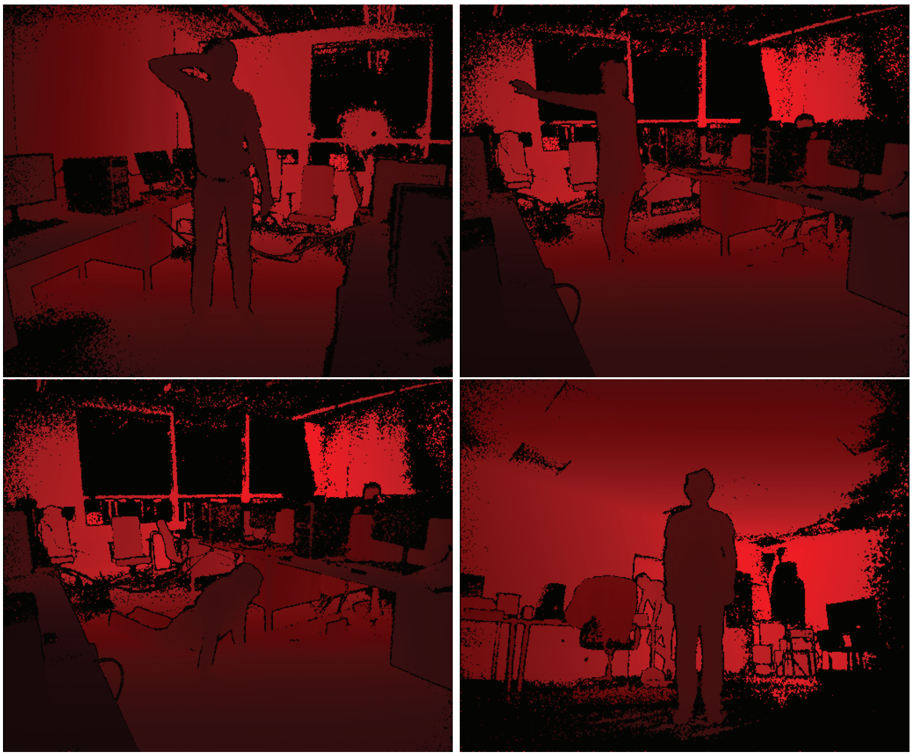


Figure 3.21. Examples of depth images used for benchmark

formed on, but the node traversal count is the same across all devices as it measures a property internal to the data structure and a related algorithm. The averages of both metrics are computed over all 1000 images.

The performance benchmark is simpler for the neural network since it solves fully-automatic segmentation tasks and does not require any human input. The model was implemented in the Python programming language using the TensorFlow library originally. Then it was saved by using the TensorFlow's own saved model format and loaded into a Java project via TensorFlow for Java. Predictions were made by using this library. A detailed implementation UML sequence diagram is shown in Figure 3.22. `AgrastSegmenter` is a class implemented for the project. The other classes are used from the library. Messages 1-2 correspond to loading the saved model from the saved model file. This only has to be done once, therefore, this part of the sequence is not added to performance benchmarks. Messages 10-17 run the Agrast-6 model. All other messages are conversions to and from data types supported by TensorFlow. Since this is required for predictions, the time taken is also added to the prediction time in the benchmark.

3.7. Multi-camera skeleton transformation and fusion

3.7.1. Skeleton transformation

Suppose we have n *Kinect* sensors $K_1, K_2, \ldots, K_n$ that all monitor the same person. Each sensor has its coordinate system CS_p, and all received skeleton points are in this coordinate system. For simplicity, let us consider two *Kinect* sensors K_1 and K_2 and two monitored reference joints J_1 and J_2. Let us also define a transformation to transform the data from CS_p to the common coordinate system CS_0 as T_p. The transformation consists of two steps:

1. Rotate the sensor coordinate space so that $x0z$ plane matches the floor plane.
2. Apply the rotation and translation transformations to match the common coordinate system.

The first step is required because each sensor may have different orientations relative to the floor plane. This orientation is detected and reported automatically by the *Kinect* sensor. Let us assume that all devices stand on the same floor plane because we already assume they monitor the same volume of space.

The floor plane's normal vector for sensor p in its coordinate system CS_p can be defined as

$$\vec{P_p} = \begin{bmatrix} A_p \\ B_p \\ C_p \end{bmatrix} \tag{3.22}$$

The goal is to rotate the view so that the vertical direction matches the direction of the y-axis. This means that transformation T_{p1} is required. This transformation could then be applied to the whole point space of the sensor. After it, the sensor stays above the floor at height D_p, which has to be subtracted to normalize the coordinate space to the ground level. Given A_p as the original sensor's p space, T_{p1} is the transformation matrix, the transformed sensor's p space A_{tp} is then

$$A_{tp} = A_p T_{p1} - (0, D_p, 0) \tag{3.23}$$

After Formula (3.23) is applied, a two-dimensional case remains to be solved to rotate and translate all sensors to match the common coordinate space. We will choose the sensor's K_1 coordinate system after Formula (3.23) as the base. First, we must find the angle between the coordinate systems of both sensors. Let us denote the vector $\vec{J_1 J_2}$ as $\vec{J}$ (see Figure 3.23). Vector $\vec{J}$ coordinates can be defined as (R_i, φ_i) in the CS_i polar coordinate space. Let us rotate vector $\vec{J}$ by angle $-\varphi_1$. The new angles between the x-axes of CS_i and $\vec{J}$ are $\varphi_i - \varphi_1$. Let us denote these angles φ_{ri}.

To find the value of φ_{ri}, we need to find the values of φ_i. According to the cosine

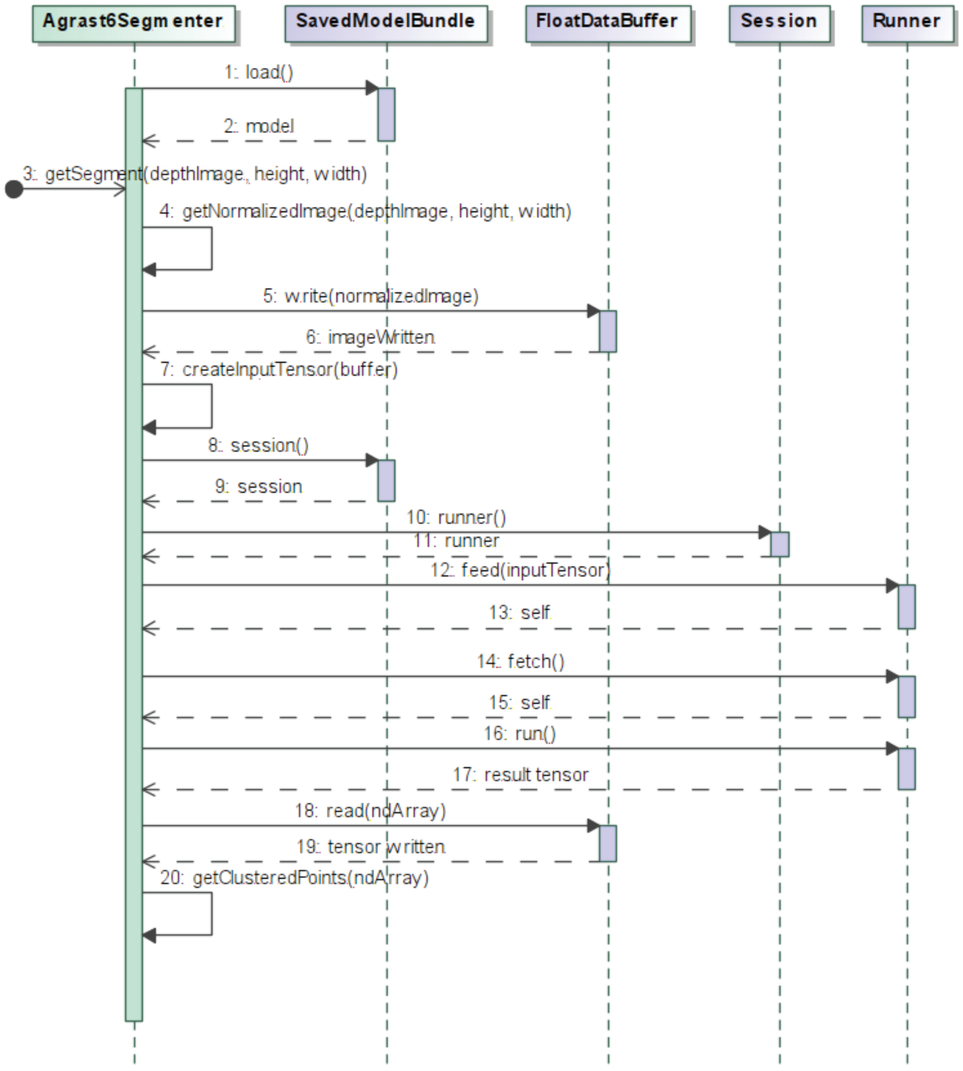


Figure 3.22. Prediction using TensorFlow for Java sequence diagram

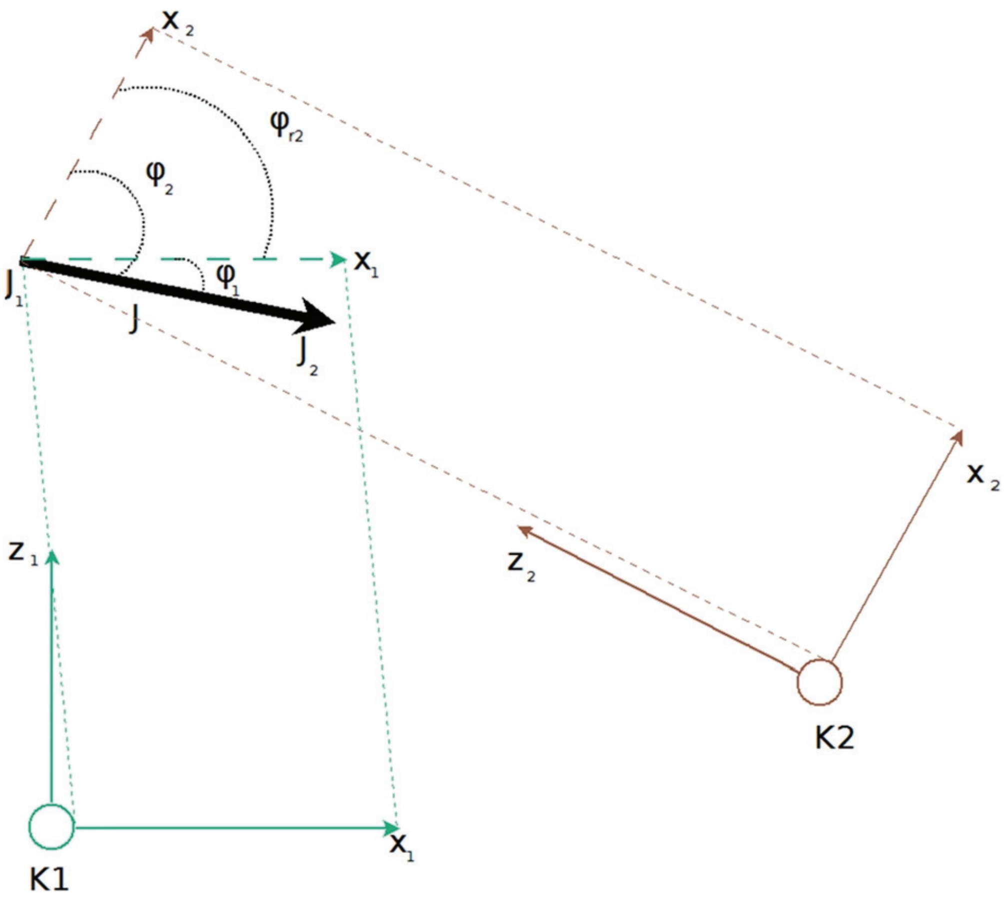


Figure 3.23. Rotation of sensor coordinate systems for data fusion

theorem, these values are

$$\varphi_i = \begin{cases} \arccos \frac{x_i}{\sqrt{x_i^2+z_i^2}}, & x_i \geq 0 \\ \arccos(2\pi - \frac{x_i}{\sqrt{x_i^2+z_i^2}}), & x_i < 0 \end{cases} \tag{3.24}$$

Suppose that the original coordinates of point J_q are $\{x_q, y_q\}$. It is then needed to rotate this by angle φ_{ri}. The resulting vector is equal to

$$\{R_q \cos(\varphi_q - \varphi_{ri}), R_q \sin(\varphi_q - \varphi_{ri})\} \tag{3.25}$$

Once the transformation T_{p1} and rotation φ_{ri} has been applied, the only transformation left to do is to move the origins of both sensors' coordinate systems to the same point. Let us use the sensor's K_1 coordinate origin as the base. Let us choose any point in space monitored by both sensors, say, J_3. The vector connecting both origins of the coordinate spaces is $\vec{K_2K_1} = \vec{K_2J_3} - \vec{K_1J_3}$. This is the vector that CS_2 must be shifted by to match CS_1. Suppose that the coordinates of J_3 are $\{x_{13}, y_{13}\}$ in CS_1 and $\{x_{23}, y_{23}\}$. Then the required transformation vector is

$$T_{22} = \{x_{23}, y_{23}\} - x_{13}, y_{13} \tag{3.26}$$

In the general case, we are going to compare sensor K_p against K_1. The required transformation is

$$T_{p2} = \{x_{p3}, y_{p3}\} - x_{13}, y_{13} \tag{3.27}$$

Thus, the final calculations to transform a set of points B_p from sensor's K_p coordinate space CS_p to sensor's K_1 coordinate space CS_1 as B_1 are as follows:

$$B_{t1} = B_pT_{p1} - [0, D_p, 0] \tag{3.28}$$

We choose any vector $\vec{J}$ of two points known by both sensors with coordinates $[x_1, y_1]$ in CS_1 and $[x_p, y_p]$ in CS_p.

$$\varphi_{rp} = \varphi_p - \varphi_1 \tag{3.29}$$

where

$$\varphi_i = \begin{cases} \arccos \frac{x_i}{\sqrt{x_i^2+z_i^2}}, & x_i \geq 0 \\ \arccos(2\pi - \frac{x_i}{\sqrt{x_i^2+z_i^2}}), & x_i < 0 \end{cases} \tag{3.30}$$

$$B_{t2} = [R_q \cos(\varphi_q - \varphi_{r2}), R_q \sin(\varphi_q - \varphi_{r2})] \tag{3.31}$$

$$R_q = \sqrt{x_q^2 + y_q^2} \tag{3.32}$$

$$T_{p2} = [x_p, 0, y_p] - [x_1, 0, y_1] \tag{3.33}$$

$$B_1 = B_{t2} + T_{p2} \tag{3.34}$$

If the sensors do not move during the monitoring, the position of the sensors does not need to be reevaluated for each calculation. It is then logical to precalculate transformation parameters φ_{rp} and T_{p2}. They can be calculated by using the same methods as described above. Then the transformation could be simplified to the following algorithm:

$$B_{t1} = B_p T_{p1} - [0, D_p, 0] \tag{3.35}$$

$$\forall q \in B_{t1} : B_{t2} = [\sqrt{x_q^2 + y_q^2}\cos(\varphi_q - \varphi_{r2}), \sqrt{x_q^2 + y_q^2}\sin(\varphi_q - \varphi_{r2})] \tag{3.36}$$

$$B_1 = B_{t2} + T_{p2} \tag{3.37}$$

This algorithm could be applied to any number of sensors. Base sensor K_1 must be chosen, and data from each other sensor K_p could be transformed to coordinate space CS_1 by using the suggested algorithm one by one. The algorithm does not require one to know the positions of the sensors in advance, so any configuration of *Kinect* sensors could be used. The only requirement is that all sensors must monitor the common volume of space.

Related research has shown that *Kinect* skeletal tracking may suffer from large fluctuations [37]. Therefore, the transformation computed from a single set of skeletons may be inaccurate. This accuracy could be improved by aggregating a buffer of skeletons while using different techniques. Low computational complexity alternatives are the mean and median values. THe buffer size is a hyperparameter of this precision increase technique. A big buffer size means that the tracking may only start after the buffer has been filled with the required values while a small buffer may suffer from large fluctuations in the data. Since the *Kinect 2* device tracks 25 joints, the transformations in Formulas (3.35)-(3.37) are computed 25 times. This means that the computational cost is low compared to any other suggested skeleton processing methods.

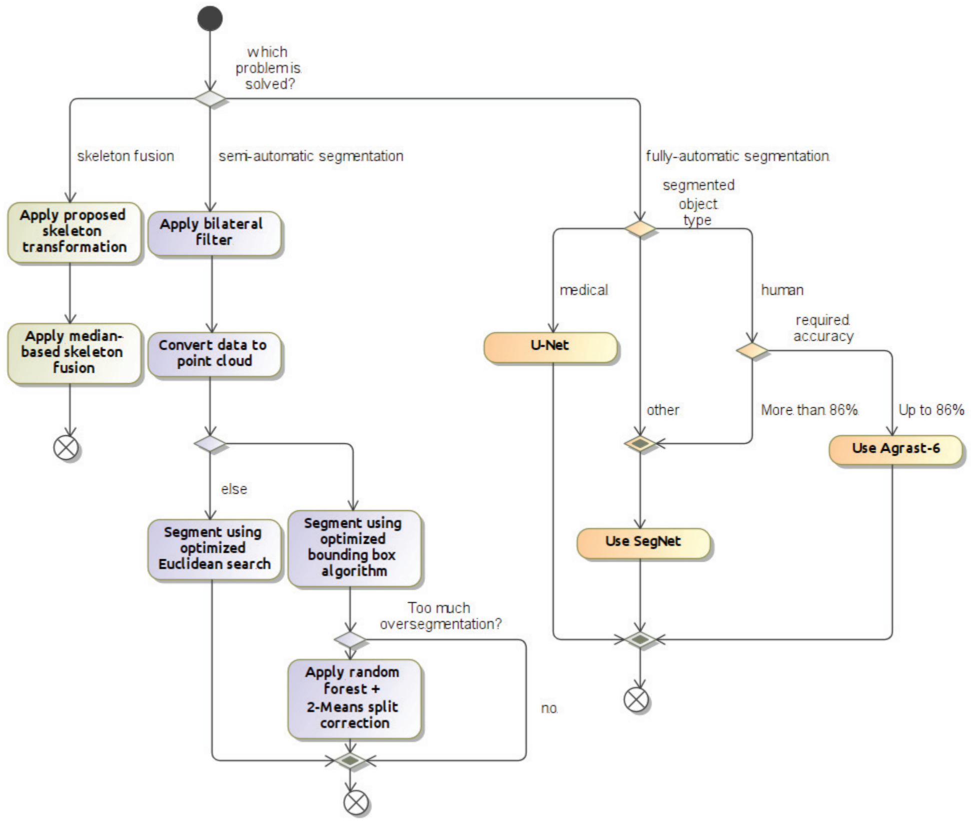
The proposed skeleton transformation algorithm is not iterative (there is a fixed amount of computations per skeleton, and the computations are not repeated), and it does not require marker objects in the scene. Complex mathematical transformations like singular value decomposition are not used, either, and only trigonometry is involved. Therefore, this approach is either less computationally expensive than the methods presented in Table 2.1, or it does not require marker objects in the scene, which makes it easier to apply in real-life scenes.

3.7.2. Skeleton fusion

Such computationally cheap methods as the average have been shown to be viable for skeleton fusion. Therefore, two methods of fusion are tested in this dissertation – the average and the median of each coordinate of each joint [105]. Average joint position $J_m = [x_m, y_m, z_m]$ is defined as

$$J_m = \left[\frac{\Sigma_{i=1}^n x_i}{n}, \frac{\Sigma_{i=1}^n y_i}{n}, \frac{\Sigma_{i=1}^n z_i}{n}\right] \tag{3.38}$$

Here, $[x_i, y_i, j_i]$ are the transformed coordinates of the joint as seen by sensor i, n is the number of sensors. Similarly, the median would be computed by using Formula



which problem is solved?
skeleton fusion
semi-automatic segmentation
fully-automatic segmentation
Apply proposed skeleton transformation
Apply median-based skeleton fusion
Apply bilateral filter
Convert data to point cloud
else
Segment using optimized Euclidean search
Segment using optimized bounding box algorithm
Too much oversegmentation?
Apply random forest + 2-Means split correction
no
segmented object type
medical
human
U-Net
required accuracy
other
More than 86%
Up to 86%
Use Agrast-6
Use SegNet

Figure 3.24. The proposed rules to select algorithms for depth or skeletal data processing

(3.38), but the averages of each coordinate over the sensors would be replaced with the median.

One more strategy used in state-of-the-art solutions is to only evaluate the joint positions that the sensor itself reports as visible (non-occluded and high confidence) [99]. This could also be combined with the previous proposition – only evaluating the average or the median from a subset of the captured coordinates. However, all these propositions require experimental evaluation of accuracy due to the uncertainty of the skeletal data provided by *Kinect*.

3.8. Proposed framework for depth segmentation and skeleton fusion

The full framework for depth segmentation and skeleton fusion is proposed based on the theoretical analysis provided in this chapter and the experimental results provided in Chapter 4. There are three distinct problems to solve – semi-automatic segmentation, fully-automatic segmentation, and skeleton fusion. They are all considered in this section, and the full framework with proposals for different scenarios is provided. The overview is provided in Figure 3.24.

First, the fully-automatic problem defined by (1.3) is best solved by machine learning methods. The best-analyzed segmentation neural networks are the *SegNet* neural network and *U-Net*. They are large architectures with over 30M trainable parameters. They also both solve a more complex task of semantic RGB segmentation. Therefore, the proposed Agrast-6 network should be used if 86% accuracy for binary segmentation is enough and the segmented object is a human. Otherwise, these larger architectures could be utilized to achieve extremely high accuracy, however, this will add a lot of inference time at a yield of a small improvement in accuracy. In that case, *U-Net* is better suited for medical images, whereas *SegNet* is better suited for other types of segmented objects.

Next, the semi-automatic problem defined by Formula (1.2) can be solved in different ways depending on the scenario. First, a bilateral filter should be used to denoise the data since segmentation has been shown to perform much better when using this filter. The depth data should then be converted to a point cloud. Next, Euclidean-clustering-based or bounding-box-based segmentation should be selected. Euclidean clustering is recommended in these scenarios where the data is complex and difficult for a human to correct. This algorithm is slow, however, it should still be useful in such scenarios since there will be fewer corrections required. In other cases, where the objects are separated in space, the bounding-box-based algorithm should be utilized since it delivers a comparable accuracy in those scenarios and works much faster. However, the proposed improvements should always be applied as they reduce the runtime of both algorithms greatly without affecting their output. It may only be worth considering in cases where the point cloud should be reused, however, resetting the internal state of the point cloud might still be faster than not using the improvements. Finally, if the bounding box was selected due to performance requirements, but under-segmentation occurs too frequently, the proposed A recursive 2-Means split algorithm with a random forest classifier for split acceptance should be used if a small performance decrease is allowed. These possibilities are all included in the software solution described in Section C.1. The user may choose between the presently mentioned algorithms depending on the data being segmented.

The proposed skeleton fusion algorithm should be used in the cases where the accuracy of a single sensor is not good enough and multiple sensors are possible to install.

3.9. Summary

This dissertation proposes the following solutions to the dissertation problems:

- An adaptation of the Euclidean clustering algorithm for segmentation;
- A bounding-box-based segmentation algorithm;
- Two improvements for Euclidean-clustering-based and bounding-box-based segmentation algorithms;
- One more improvement for the bounding-box-based segmentation algorithm only;
- A recursive 2-Means split algorithm with a random forest classifier for split acceptance for under-segmentation reduction;
- *Agrast-6* convolutional neural network architecture;

- *Kinect* skeletal data transformation and fusion algorithm.

First, to conduct the segmentation-related research, two datasets have been captured. These datasets consist of short depth video sequences, containing people in a variety of poses. They have been captured by using three *Kinect* cameras simultaneously. Next, a novel segmentation evaluation metric, cross-set intersection, is introduced. Its analysis shows that it offers advantages over the currently available metrics:

- In cases where one of the segmentation output and the ground truth is a subset of the other, cross-set intersection, unlike the Dice's score, has an easy-to-interpret numeric value (it is the ratio of the sizes).
- It is faster to compute than Jaccard score because the union of both sets does to be computed.
- Its value never exceeds the ratio between the sizes of the set intersection and either of the sets, for which the intersection was computed, thereby preventing relatively high numeric values for low segmentation accuracies.
- It involves an extra penalty compared to both Dice and Jaccard metrics in cases where both under-segmentation and over-segmentation may occur.

These properties make the cross-set intersection a good choice for segmentation quality evaluation.

The semi-automatic segmentation problem can be solved by adapting an already existing Euclidean clustering algorithm. The original algorithm is meant for clustering, therefore, it selects a point from the point cloud and constructs a cluster by repeatedly adding all the points that are closer to at least one point of the constructed cluster than the predefined distance. When no more points can be added, an unprocessed point is selected, and a new cluster is constructed. The algorithm terminates when all points belong to the cluster. If point selection is replaced with a human-selected point and only one cluster is selected, this becomes a segmentation algorithm.

Unfortunately, this algorithm is not efficient because the algorithm has its state (thefound clusters) which is tracked separately from the point cloud. The nature of the algorithm allows the same point to be found multiple times, however, it has to be a part of a single cluster. This leads to the tracking of the found points and then checking if the point already belongs to some cluster, which degrades the performance. Another performance-related issue is the definition of the object shape – it is a super-set of spheres around each point that belongs to the object. Checking if a point belongs to such a shape is computationally expensive if the object consists of many points.

The latter issue is proposed to be solved by simplifying the object-defining shape. A bounding box is a good candidate because its shape is simple to define and update, and it is very easy to check if a point is inside of the bounding box. The former issue, on the other hand, can be solved by moving the state of the algorithm inside the three-dimensional binary tree. Instead of tracking the processed points separately, tree nodes can be marked as removed. This has further advantages as, instead of finding potentially too many points, checking which points should be discarded, and then discarding the points, a flag is checked and flipped. Even more, if a whole branch of the tree belongs to a segment, it can be marked as removed, which means that it can be skipped completely as if it has been cut from the tree, but without actually modifying it. One final performance improvement is possible for the bounding-box-based segmentation. If the original Euclidean clustering is followed and adapted to use bounding boxes, the bounding box is defined, all points inside of the box are found, the bounding box is expanded so that the minimum distance from any contained point to any side of the box is not lower than the predefined minimum, and the process is repeated. However, the bounding box can be expanded after adding each point rather than after a full pass over the tree, which allows it to grow much faster and reduce the number of passes. The datasets mentioned above have been segmented by using software that utilizes the bounding-box-based segmentation algorithm.

Unfortunately, the bounding-box-based segmentation algorithm is more prone to under-segmentation as it tends to include parts of the background, and not only the object, more often than the Euclidean-clustering-based approach. However, the Euclidean-clustering-based segmentation is not immune from this problem, either. It can be solved by recursively repeating the 2-Means split of the segmentation output and potentially discarding one of the segments. The output is split into parts by using the K-Means algorithm where $k = 2$. Then, 8 metrics are computed, specifically, the distances between each combination of the two centroids and three clusters (both sub-clusters and the original super-cluster), as well as the sizes of both sub-clusters. A random forest classifier, trained on the values of those metrics, then decides if the split improves the segmentation accuracy or not. The splits are repeated until the accuracy has stopped improving.

The fully-automatic segmentation problem can be solved by using the already existing neural networks for multi-class RGB segmentation, however, binary depth segmentation is a simpler task that could be solved by using a smaller architecture. Therefore, the *Agrast-6* architecture is proposed. It is a smaller version of the *SegNet* neural network (it has 13 layers instead of 26, and the layer depth is lower, but it keeps the symmetry between the encoder and the decoder) without the skip connections. A bounded ReLU layer is used instead of a Softmax layer because the pixel labels are binary. This architecture has only 1.25M trainable parameters. The network has been trained by using segmented datasets.

The skeleton fusion problem can be solved by using the proposed geometrical algorithm. All the coordinate systems of *Kinect* sensors are transformed to match the floor plane. A rigid human body part (two joints) is selected, and each sensor's viewing angle is estimated by using the coordinates of the vector representing this body part. Then all views of the sensors can be rotated to match the first sensor. Skeletons can then be fused by using the median or mean coordinates.

4. EXPERIMENTAL RESULTS

This chapter consists of an experimental evaluation of the algorithms proposed in the previous chapter.

The same datasets were used for most of the research and are therefore presented first in Section 4.1.

Section 4.2 presents in-depth analysis of the proposed bounding-box-based segmentation algorithms and performance improvements and verifies if the proposed algorithms can reduce the total human-supervised segmentation time. First, noise reduction algorithms are compared through the lens of human body segmentation – their impact on the final accuracy is measured as well as their runtime. Next, all the proposed bounding-box-based search performance improvements are evaluated separately and in combination and compared to Euclidean-clustering-based segmentation. Some performance improvements are also applicable to Euclidean clustering, and their impact on this algorithm is measured as well. It is shown in this subsection that the performance improvements give a huge performance boost for binary segmentation. Next, the accuracy of Euclidean segmentation and bounding box segmentation is analyzed in multiple ways as the total average cross-set intersection is compared, the results are also dissected by the camera angle and pose, accuracy histograms are provided and qualitative error analysis is conducted. In addition to that, sex-wise accuracy measurement is also provided. Finally, savings of human work are estimated, which is one of the key focal points of this dissertation.

Section 4.3 provides in-depth analysis of the proposed recursive 2-Means split algorithm with a random forest classifier for split acceptance to reduce under-segmentation errors. The analysis follows a similar structure as the bounding box segmentation. First, performance analysis is provided. Extra work required by this algorithm is analyzed. Next, detailed accuracy analysis follows with the same structure as the bounding box segmentation accuracy analysis. Finally, an estimate for the total segmentation time cost is presented. This section tests the statement that the proposed split, metrics, and classifier may reduce under-segmentation.

The machine learning architecture Agrast-6 is analyzed in Section 4.4. First, the training progress is overviewed both qualitatively and quantitatively to verify that the network is learning. Next, image processing analysis is provided to inspect how Agrast-6 processes an image and how the features are encoded and decoded. This is followed by performance measurements to estimate the prediction time. Accuracy analysis is also presented in the same format as in other chapters. This section evaluates the viability of the proposed smaller architecture and the hypothesis that an abridged architecture may solve the binary human body segmentation task with accuracy comparable to state-of-the-art solutions for RGB segmentation.

Finally, Section 4.5 provides a comparison of accuracy of different algorithm configurations as well as performance measurements.

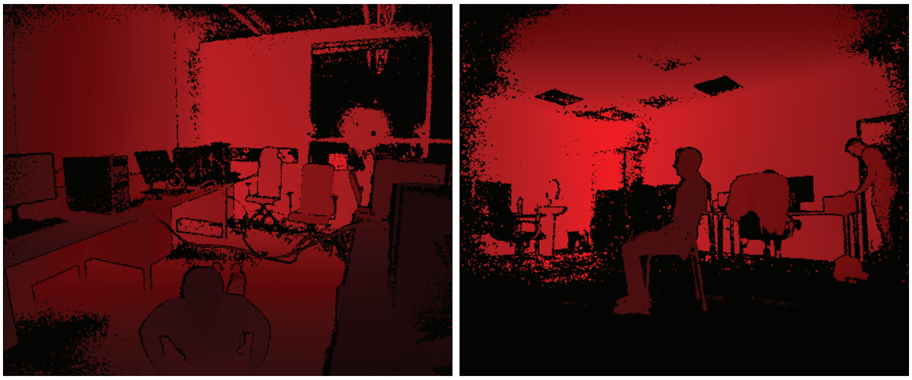


Figure 4.1. Example depth frames. Left – complex dataset, right – simple dataset

Table 4.1. Details of captured datasets

Dataset name	Complex	Simple
Number of recordings	674	266
Number of frames	193k	69k
Number of viewing angles	3	3
Environment	More cluttered classroom	Less cluttered classroom
Number of participants	7	40
Participants by sex	5 males, 2 females	20 males, 20 females
Number of poses	30	2

4.1. Datasets

Two datasets were captured. The first dataset, further called ‘complex’, is 7 people in 30 different poses (various arm positions, standing, squatting, lying on the ground, etc.). It consists of 674 recordings (193k depth images) from three different angles (front, back at a small angle, and side at an angle). The surroundings are relatively cluttered, the subjects had just enough space to perform their poses. The subjects were almost static during the recording process which was about 10 seconds per recording. The second dataset, further called ‘simple’, is 40 people in standing and sitting poses (the latter include a chair). It consists of 266 recordings (69k depth images), also from three angles (front, 120, and 240-degree angles). The surroundings were much less cluttered with a couple of meters of free space around the subject. Example depth images from both datasets are shown in Figure 4.1. The datasets consist of static scenes due to the limitations uncovered by related research [69]. A detailed comparison of both datasets is provided in Table 4.1. A list of poses with their corresponding images is provided in Appendix C.

4.2. Semi-automatic segmentation

4.2.1. Data formats

Both depth maps and 3D binary search trees have properties useful for depth data processing. Therefore, both are utilized for a variety of purposes of this research. The depth map is first utilized for noise reduction, then a 3D binary search tree is constructed from a denoised depth map. Since the *Kinect 2* data stream is 512x424 pixels, 217088 pixels are stored in total. The depth values correspond to short integers, however, the RealSense depth pixels require coordinate remapping. Depth values may be non-integer after remapping, therefore, it is reasonable to store them as double floating point values. Since one double value takes 8 bytes of memory, the depth map takes 1.66 MB of memory. A binary search tree is a larger structure. It is implemented as a structure of multiple values:

- `x`, `y`, and `z` coordinates, 8 bytes each make 24 bytes total;
- `x` and `y` coordinates in the original depth map for faster depth map reconstruction, 2 bytes each constitute 4 bytes total;
- references to the left and right tree branches, 8 bytes on a 64-bit system, 16 bytes total;
- Boolean flags for removed and removed children, required for performance improvements, 1 byte each, 2 bytes total;
- the current depth of the tree, 1 byte

Thus, memory consumption is at least 47 bytes per point, or ~9.73 MB per frame. The resulting data – the binary masks – are also stored as a one-channel image because it takes much less space than a search tree. Both the input depth stream and the output binary masks are serialized to and from the disk by using *Protobuf* library[1].

The data structure is depicted as a UML class diagram in Figure 4.2. `TreeNode` is a class that represents a tree structure. Since this is a binary tree, it has the left and right children nodes which can have a null value for leaf nodes. However, no link to the parent is required for the implementation. This is achieved by using a recursive directed association between the objects of `TreeNode`. Each `TreeNode` also contains a `Point` instance that holds the actual data contained by the tree. A reference to the root node of the tree is used to process it.

While the memory consumption of a binary tree is already at 2.9 GB per 10 second long frame sequence, octrees have not been considered since the memory consumption would have been even larger.

4.2.2. Experimental noise reduction analysis

Four variants of noise reduction have been tested – no noise reduction, median filter with window size 3, the Gaussian filter with $\sigma = 0.8$, and the bilateral filter with $\sigma_R = \sigma_D = 0.8$. All filters use their custom implementations. The filters were run against both datasets by selecting every 10th image. The accuracy was measured separately for both datasets. The runtime does not depend on the data, so it was averaged throughout all images. All images were processed by the same segmentation algorithm. Thus, only the 'reduce noise' action from Figure 3.7 was changed between the experiments.

The results are shown in detail in Table 4.2. The table shows the first quartile (Q1), the median (Q2), the third quartile (Q3), and the average accuracies per dataset

[1]https://developers.google.com/protocol-buffers/

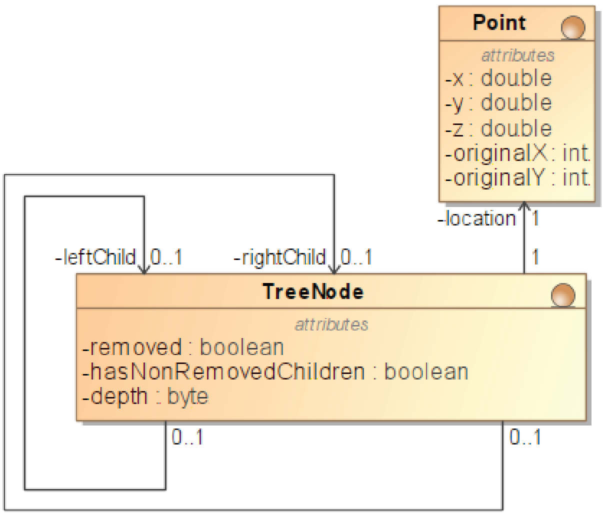


Figure 4.2. Binary search tree data model

type. The simple dataset is processed by a bilateral filter much more accurately as its accuracy average reaches 81.6%, which is much higher that the runner-up median filter, whereas no filter at all yields 67.7% accuracy. However, Q1 results suggest that if the algorithm performance is low, no filter is going to change it. Complex dataset results tell a similar story. Q1 results are similar across the board, while the bilateral filter beats all other alternatives by a large margin.

The accuracy results suggest that the median filter has little impact on the output. The Gaussian filter blurs the edges of the objects to the point where the boundaries of these objects cannot be used for segmentation anymore. The bilateral filter, on the other hand, enhances the edges of the object and therefore improves segmentation quality.

The performance results show that the bilateral filter is the most complex to compute. Since the bilateral filter is a Gaussian filter with a point similarity weight, it is apparent that the similarity weight is the difficult part of the computations. It takes 77% of the whole filtering time. The median filter is slower than Gaussian because it involves sorting an array of neighbor pixels, which is more computationally expensive than just applying the Gaussian kernel. No filter takes 1.4 ms because this is the time required to convert a flat array into a 2D array. The results show that a bilateral filter should always be used, except when the execution time is critical to the point that 32 ms savings are worth a lower performance – no filter should be used in that case.

Figure 4.3 provides a qualitative comparison of the filters. This image is selected from the simple dataset and contains two problems. The subject was wearing black jeans which absorbed signals emitted by the *Kinect 2* sensor, and therefore the depth values are present only in a part of the image. The image is also noisy around the floor near the subject's legs. The first sub-figure shows some fine-grained noise on the trousers. The median filter reduces this noise by removing small blobs of the

Table 4.2. Denoise function accuracy impact and performance comparison

Denoise function	Dataset	Q1	Q2	Q3	Average	Runtime, ms
None	Simple	72.0%	78.0%	80.7%	67.7%	**1.4**
	Complex	5.9%	7.8%	10.8%	10.5%	
Median	Simple	72.9%	77.9%	80.4%	67.7%	16.8
	Complex	6.2%	8.0%	11.0%	12.2%	
Gaussian	Simple	2.4%	2.8%	4.1%	3.3%	7.6
	Complex	4.5%	5.7%	8.2%	6.4%	
Bilateral	Simple	**92.6%**	**97.4%**	**100.0%**	**81.6%**	33.7
	Complex	**7.3%**	**11.0%**	**16.4%**	**21.2%**	

actual depth values. The Gaussian filter does not do this and blurs the whole image equally. However, the bilateral filter produced the most interesting effect. It completely removed the depth values on the trousers, which would be useful when training a neural network; hence, the depth values are not there, however, the legs are still visually visible. However, this completely prevents semi-automatic segmentation on the legs. Another effect is a very distinct border around the human body and other objects. This new boundary makes it much easier for a segmenter to find the edges of the object. Therefore, qualitatively, a bilateral filter seems to work best for binary segmentation, which agrees with the experimental qualitative results.

In addition to that, the bilateral filter 'draws' a boundary around the object which increases the distance between the object pixels and the surrounding pixels. This artificial boundary helps prevent the condition defined in Formula (3.8). h_o is artificially increased, and the chance of it being lower than h_{max} is reduced. This also helps to increase the quality of Euclidean segmentation.

4.2.3. Qualitative analysis of watershed segmentation

The classic watershed segmentation approach was implemented to process depth data by using the OpenCV library [344]. However, since the data was noisy in the depth frames coming from depth-sensing devices, the clusters generated by the watershed were too fine. Different image noise reduction techniques were applied, and it was observed qualitatively that erosion is the only preprocessing step that improves the clustering, however, the results are acceptable only in a small subset of data. Some output examples are provided in Figure 4.4. It is clear that the watershed is sometimes capable of extracting the human body as a single cluster, however, some noise is also included, or the cluster has many holes that have to be fixed by a post-processing step. This confirms the problems stated in related research [228] – this algorithm has troubles with noisy data. It was decided not to analyze this algorithm any further since not only the segmentation accuracy is low, but it would also require the detection of the cluster that is the human.

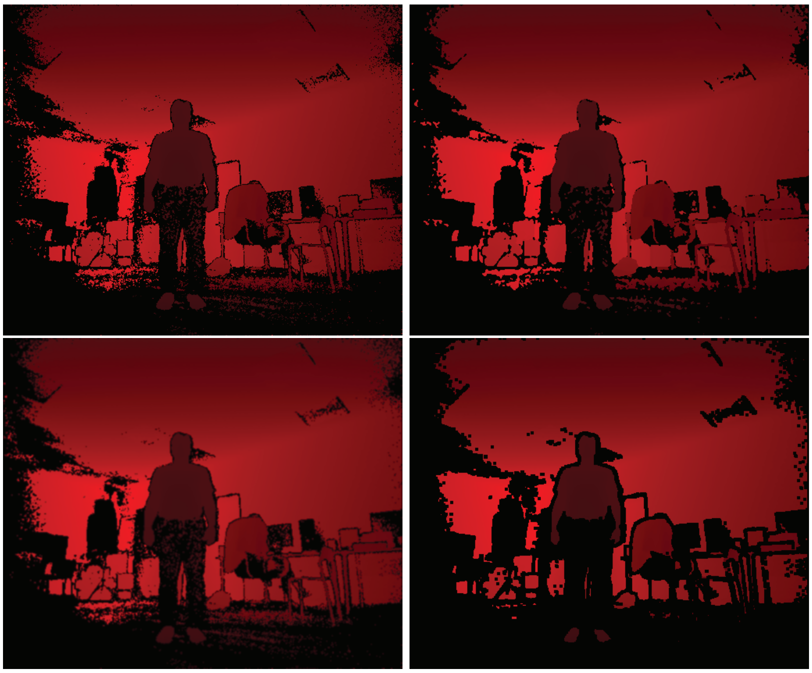


Figure 4.3. Filter visual comparison: no filter, median filter, Gaussian filter, and bilateral filter, respectively

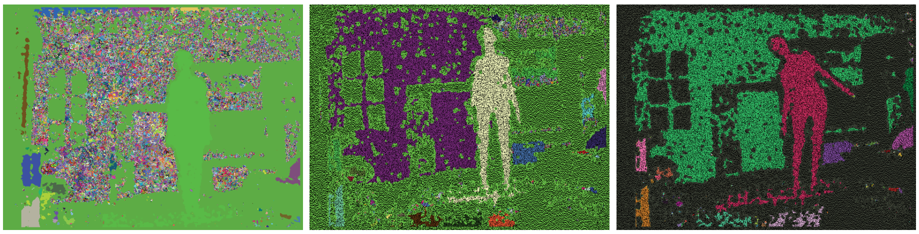


Figure 4.4. Watershed-generated clusters

4.2.4. Experimental analysis of point cloud processing performance improvements

4.2.4.1. Benchmark results

Experimental analysis has shown that the basic bounding-box-based segmentation algorithm with node removal is a lot faster than the option with no performance improvements. The difference in runtime is 23.4 times, whereas the difference in the node traversal count is 8.76 times. These results suggest that the proposal improves the results in two distinct ways. First, the node traversal count is reduced by a large margin. This translates into a better runtime. Second, since the removed nodes are stored inside the tree itself, there is no need to keep track of the removed nodes separately, which also consumes a very large portion of computation time. This is evident for all results because any algorithm that tracks the segmentation state externally is slower than any algorithm that tracks the state internally.

Auto-expanding bounding box improvements have shown a great effect on performance. This variant of the algorithm does not move the state inside the tree, and the performance increase is purely due to a reduced iteration count. The runtime has been reduced by 11.7 times compared to the basic bounding-box-based search algorithm, whereas node traversals have been reduced by 15.1 times. This shows that the improvements of the auto-expanding bounding box reduces the search complexity by an order of magnitude. The node traversal count reduction is achieved by an even larger margin than by utilizing node removal improvements, however, the runtime still suffers from extra work to manage the external state.

The branch-cutting improvement also has a positive outcome in terms of performance. It reduces the runtime of the algorithm further by 13.5%, and the node traversal count by 50.5%. The node traversal count improvement is significant, and it allows to terminate the search earlier, thereby saving half of the tree node traversals required otherwise. This shows that this improvement is worth implementing since the completely removed subtrees are either common or get large. However, there is a small extra runtime cost to check and update the removed subtrees, therefore, the time savings are not proportional to the node traversal count reduction.

Switching between the left-to-right tree traversal and the root-first tree traversal does not have much impact in real-world scenarios. It is unexpected that, while the differences in the runtime are small, the node traversal count is always lower in the left-to-right traversal. However, the left-to-right tree traversal requires a little extra work. The traversal of the search tree eliminates all branches that are known to be outside the search range. If the root node is visited before its children, both lower and upper boundaries can be computed together. However, if an expanding bounding box is used with the left-to-right traversal, the minimum boundary must be checked before the bounding box expansion, and the maximum boundary after the expansion. K-dimensional trees split their data by alternating dimensions at each level, therefore, different bounding box properties have to be checked. For example, if the depth is 0 (the root node), the x coordinate of a 3D point has to be checked, but the y coordinate has to be checked at depth 1. If the left-to-right traversal is used, this check has to be performed twice, which adds extra work. This extra work, however, is not compensated by a lower tree node visit count. Therefore, it is not worth using the left-to-right node traversal.

The hypothesis that the improvements are orthogonal in the sense that they are independent (presented in Section 3.4.1.5) seems to hold, however, the exact answer cannot be found in the performance benchmark. Combining two improvements always gives a better runtime than using only one of those improvements. Therefore, the best result is achieved when all improvements are applied. For example, no improvements yields the worst performance of 1.4 seconds per frame. Applying the node removal reduces the runtime to 61 ms (about 23 times faster), and applying the auto-expanding bounding box takes 122 ms per frame (about 11.5 times faster). Combining both improvements reduces the runtime to 17 ms per frame (83 times faster). If the improvements were truly independent, the expected increase in performance should be $23 \times 11.5 = 264.5$. The fact that the observed number is lower can lead to two possible explanations (or a combination of them). First, the improvements, while being orthogonal in terms of the analyzed properties of the tree, cut the runtime in the same ways, or the runtime cuts impact each other. Most probably, the expanding bounding box reduces the number of iterations over the tree by visiting more points during one iteration. Since the number of points increases, more points that do not fit into the bounding box are visited, and the effect of marking points as removed works on a smaller proportion of the points. Moreover, all points are not marked as removed in the first iteration, but the expanding bounding box makes the first iteration longer. Second, they may optimize the same part of the code while not optimizing the other part at all. Both improvements optimize the method that traverses the tree, but none of the algorithms optimizes the part where points are concatenated into a single collection between the iterations. Therefore, it is not entirely correct to just multiply the performance increases – this is the potential performance increase *for the optimized part*, not for the whole algorithm.

The Euclidean-clustering-based segmentation is used as a baseline. It has been re-implemented in the same software solution according to the UML diagram depicted in Figure 3.8 which was, in turn, adapted from the Euclidean clustering algorithm provided by Rusu [49]. It operates the same tree data structure as the other suggested algorithms, therefore, it removes performance differences due to different platforms. In addition to that, an original Euclidean clustering algorithm from the PCL library (implemented in C++) has been benchmarked, however, there is no way to measure the tree node traversal count by using this library. The Euclidean clustering algorithm has an upside and a downside compared to the unoptimized bounding-box-based algorithm. The advantage is the less state to keep track of and the simpler state updates. The Euclidean-clustering-based segmentation only operates one point at a time, therefore, the runtime of a single search iteration is almost always close to $O(\log n)$ complexity. The downside is that it performs one search for each found point. Optimizations of marking nodes as removed and marking branches as removed work with this algorithm as well, and the runtime reduction is 4.1 times. The effect is not as large as with

Table 4.3. Algorithm runtime and node traversal count comparison

Algorithm	Runtime, ms	Node traversals, M
Euclidean clustering (baseline, original implementation)	1233	-
Euclidean clustering (baseline)	2326	194
Basic bounding box	1426	34
Euclidean clustering with branch cutting	561	79.6
Basic expanding bounding box	122	2.25
Basic expanding bounding box LTR	119	1.83
Bounding box with node removal	60.9	3.88
Bounding box with branch cutting	52.7	1.92
Expanding bounding box with node removal LTR	20.4	1.95
Expanding bounding box with node removal	17.1	2.50
Expanding bounding box with branch cutting LTR	6.52	**0.284**
Expanding bounding box with branch cutting	**6.33**	0.311

the bounding-box-based segmentation because the points are removed much slower than with bounding boxes – that is, if the bounding box is large in two dimensions, the increase in the third dimension adds a relatively large volume of the search space. Euclidean-clustering-based segmentation does not have this property as its scan range is very small compared to the large volume of the bounding boxes. While it includes a few points per iteration, few points are marked as removed per iteration. However, the speedup is still very nice.

A detailed comparison of all runtimes and tree node traversal counts is shown in Table 4.3.

The results show that the suggested set of algorithms and their improvements make the segmentation about whopping **367 times faster** than the baseline algorithm when implemented in the Java programming language, and even the Java implementation is faster than the original C++ implementation by a massive margin – **195 times**. As expected, the Java implementation is about 2 times slower than the C++ implementation of the baseline algorithm. The benchmark also shows an overwhelming **624 times fewer tree node visits** to fully segment a point cloud. While it should be noted that the algorithms produce different results, and this could impact the performance to some extent, the solution is still 2-3 orders of magnitude faster with real-life point clouds than with the regular Euclidean clustering.

4.2.5. Experimental analysis of point cloud processing accuracy

4.2.5.1. Accuracy benchmark results

Overview

Four variants of segmentation have been tested for accuracy – Euclidean-clustering-based segmentation (baseline), a bounding box with branch cutting, an auto-expanding bounding box with branch cutting, and an auto-expanding bounding box with branch cutting with the left-to-right traversal. All other variations yield the same segmentation output as one of the four selected algorithms, or the variation is irrelevant for this research.

Table 4.4. Average accuracy comparison

Algorithm	Simple dataset accuracy	Complex dataset accuracy
Euclidean clustering (baseline)	79.8%	48.1%
Bounding box with branch cutting	82.1%	33.4%
Expanding bounding box with branch cutting LTR	82.1%	33.4%
Expanding bounding box with branch cutting	82.1%	33.4%

The benchmark has shown that the bounding-box-based search is similar in accuracy to Euclidean-clustering-based segmentation for the simple dataset. The humans were isolated in their environment, which means that the bounding box volume is not likely to include extra objects – they are simply far away from the person. The difference in the cross-set intersection accuracy was relatively small – 2.3 percentage points in favor of the bounding box segmentation. This shows that both algorithms are suitable for the segmentation of simple scenes. The complex dataset, however, tells a different story. The Euclidean-clustering-based segmentation was much more accurate in this regard – the difference was 14.7 percentage points. It was expected to be more accurate in a cluttered environment due to the much finer-grained body of the object. The difference is sizeable, which makes the Euclidean-clustering-based segmentation feasible where a much slower performance is a good trade-off for the extra accuracy.

An interesting finding is that all the three variations of the analyzed bounding-box-based algorithm yielded almost the same results. The differences are within 0.05 percentage points. It is safe to assume that the difference is negligible, and there is no scenario where the slower algorithms give any benefit. This shows that, while the type of the bounding box expansion has a great effect on the performance, it does not affect the accuracy at all.

The summary comparison is detailed in Table 4.4.

Detailed analysis of the results

Since all the bounding-box-based algorithm variations showed the same accuracy, only the results of the Euclidean-clustering-based segmentation and the expanding bounding box with all the improvements shall be analyzed further.

First, let us analyze the results from the simple dataset. They are detailed in Figure 4.5. The results are dissected by the viewing angle and the human pose. This dataset was captured by using three *Kinect* cameras – one was facing the face (the front side) of the human, the second was at an angle between the right side and the back (135 degrees angle), and the final one was looking from the left side. The people were either standing or sitting. The number of samples for each pose-angle pair is the same.

The box plots reveal that the viewing angle has a lot of impact on the accuracy of both algorithms. The front camera view was segmented with the best accuracy. The front view for the standing pose was segmented so accurately by both algorithms that anything below 99.9% match is considered an outlier, which is also seen on the box plot. The bounding-box-based segmentation offered a slightly higher accuracy. Q1 accuracy (the lower quartile) was still at 97.2% for the standing and sitting poses combined.

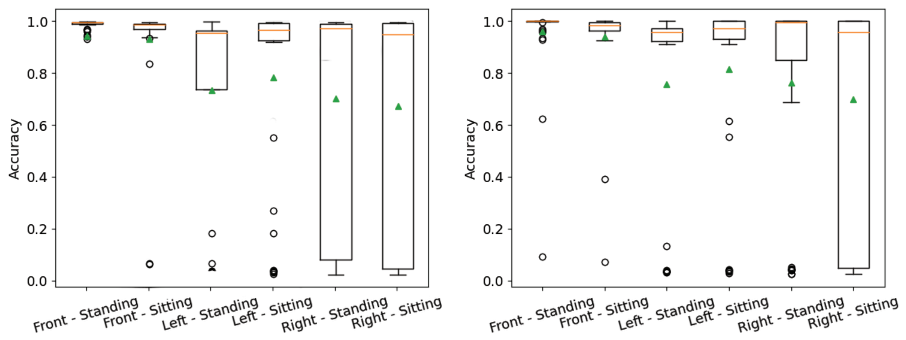


Figure 4.5. Simple dataset accuracy comparison. Left – Euclidean-clustering-based segmentation, right – bounding-box-based segmentation

Other angles were more difficult for both algorithms to process accurately. Both left and right angles had similar average accuracy in all cases, however, the right angle had more frames where the accuracy was very low. This can be seen by Q1 values, and they are below the 20% mark for the Euclidean-clustering-based segmentation for both poses from the right angle, but the bounding box has a much higher Q1 accuracy. Nevertheless, there are still numerous outliers in the sub-20% area for this case as well. The standing pose was also easier to segment for both algorithms than the sitting pose. The point cloud for a sitting person is more complex in the sense of the surface shape visible by a depth sensor. There is a near-90-degree angle in two locations (the knee and the hip) which may separate the parts of the body by occluding it via wrinkles of clothing. Body self-occlusion is also more likely in a sitting pose.

The average and median values are close for both algorithms. If the data for both poses is combined and Q1, Q2, and Q3 values are computed per camera angle, the largest difference is the right angle Q2 value (median) – it is 95.9% for Euclidean-clustering-based and 99.4% for the bounding-box-based segmentation. All other differences are less than one percentage point.

These results suggest that the bounding-box-based algorithm is better suited for datasets of this type. The accuracy difference is even in favor of the bounding-box-based algorithm, however, it is small. However, as shown in the previous section, the bounding-box-based algorithm with all improvements is so much faster, that all accuracy differences are small in comparison to the added efficiency. Therefore, for simple data, where the objects are separated in space and the environment is not cluttered, the bounding-box-based search is a better alternative by all measures.

Complex dataset results are different. They are shown in Figure 4.6. The back side was processed with a similar accuracy, the differences between the medians, averages, and quartile values are small. This view was at an almost-180-degrees angle – the back of the human was clearly visible. However, the poses were more complex, and this pushed Q1 values for both algorithms to around 20%. The front side was at around 30-degrees angle, and it was processed much more accurately by the Euclidean-clustering-based segmentation. The Q1 value for the Euclidean-clustering-

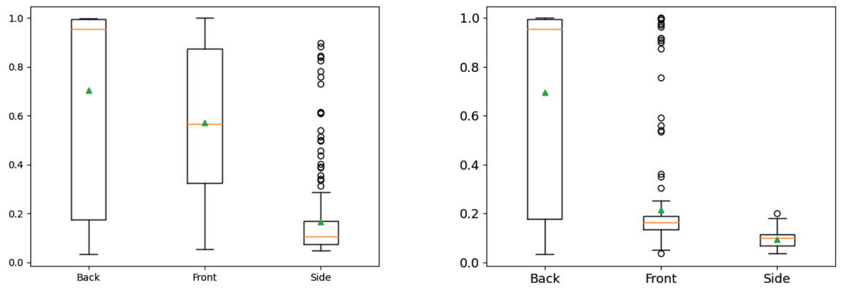


Figure 4.6. Complex dataset accuracy comparison. Left – Euclidean-clustering-based, right – bounding-box-based segmentation

based segmentation was higher than the Q3 value for the bounding-box-based segmentation. While some frames were segmented with better accuracy, sub-20% Q3 shows that the segmentation quality was generally poor. The biggest difference is for the side view. Both algorithms did not do very well with Q3 values lower than 20%. However, the Euclidean-clustering-based segmentation was able to segment at least some frames with reasonable accuracy – there are outliers with over 75% accuracy. The bounding-box-based segmentation algorithm failed much worse with only one frame slightly crossing the 20% accuracy mark.

The viewing angle had a similar impact on both algorithms as well. However, the cluttered nature of the scene negatively impacted the bounding-box-based algorithm. It was also observed for the Euclidean-clustering-based segmentation to a lesser extent. However, the differences in accuracy are large for the front and side views, as the bounding-box-based algorithm seems to be unusable at all under these circumstances. The side view was too complex for the Euclidean-clustering-based segmentation as well, except for a few cases.

These results show that the applicability of the Euclidean-clustering-based vs. bounding-box-based segmentation is debatable. The back side results suggest that the accuracy in this scenario is similar, and the bounding box performance has a clear advantage. However, this is not the case for the front and side views. The front view is processed by the Euclidean-clustering-based segmentation with a noticeable error, however, the results are still usable in most cases. If the application is semi-automatic segmentation, this output is easy to fix. The bounding box segments some frames with a good accuracy, but generally fails. Therefore, the Euclidean-clustering-based segmentation is worth considering if a longer processing time is acceptable. The side view is too complex for both algorithms, and the only difference is computing for a short time and then failing, or computing for a longer time and then failing.

This analysis shows that the bounding box performs worse than the Euclidean-clustering-based segmentation in one dissection of the data – which is the side view of the complex dataset. It means that cluttered environments are processed more accurately by the Euclidean-clustering-based segmentation. In other cases, either both

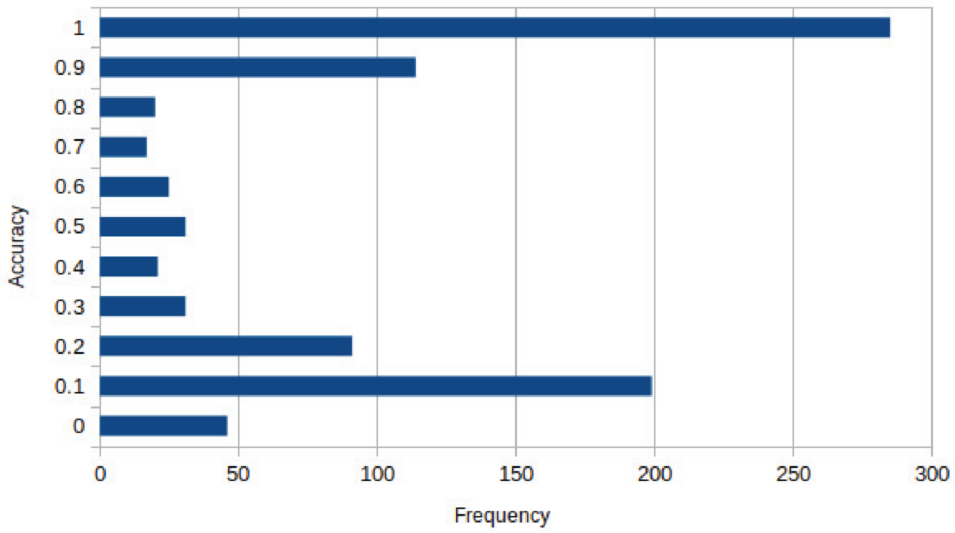


Figure 4.7. Euclidean-clustering-based segmentation accuracy histogram

algorithms fail, or the accuracy is comparable, but since the bounding-box-based algorithm has much better performance, it should be used for segmentation where the scene is involved where the object is clearly separated from the surroundings.

Accuracy histogram analysis

Figures 4.7 and 4.8 show the cross-section intersection value occurrence frequencies for the Euclidean-clustering-based and bounding-box-based segmentation, respectively. These results show that the bounding box segmentation leans toward either a good segmentation accuracy (over 90%), or a bad segmentation accuracy (under 30%). The Euclidean-clustering-based segmentation yielded significantly more segmentations with a medium accuracy (30%-80%). This reflects the results presented in Figures 4.5 and 4.6. The Euclidean-clustering-based segmentation was more accurate for the complex dataset, however, the results were mostly around the mid-range, which can be seen in the histogram as well.

These results suggest that the bounding-box-based segmentation is more sensitive. If the bounding box captures some pixels of another object, it includes that object, and the box expands much further than the boundaries of the original object. In the case of under-segmentation, accuracy of 20% means that the segmentation output contains 5 times more pixels than the original object. The Euclidean-clustering-based segmentation tends to be less sensitive to this type of error, however, it is not completely prone to it. Both algorithms had approximately the same amount of frames with over 90% accuracy. This suggests that the bounding box may be more useful if the segmentation has to be precise or thrown away. However, if partially correct segmentation is acceptable at a performance penalty, the Euclidean-clustering-based segmentation should be used instead.

Qualitative error analysis

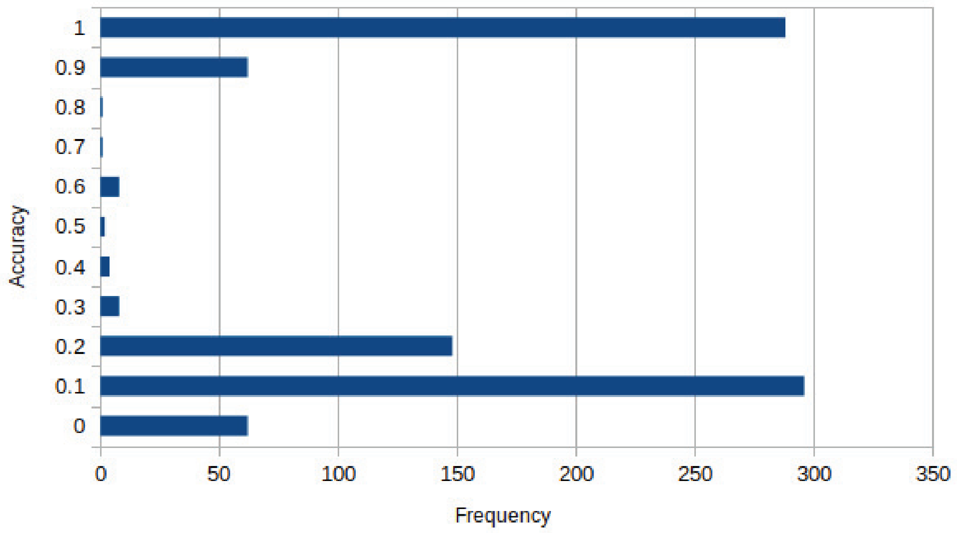


Figure 4.8. Bounding box accuracy histogram

Figure 4.9 shows common mistakes made by the Euclidean-clustering-based segmentation. The shown frames feature 4, 5, 8, 9, 91, and 94% cross-set intersection accuracy. The pictures are color-coded: the red color is the segmentation output, green is the ground truth, and yellow is their intersection. All observed errors with up to 79% accuracy were under-segmentation errors. This is visible in the first four frames – the human body is mostly yellow, which means that the Euclidean-clustering-based segmentation captured the body and a large part of the background. The cross-set intersection of the first image is 4%, and only a small part of the body was not captured. This means that the ground truth intersection with the segmentation output is almost equal to the ground truth itself. Therefore, the red under-segmented area is almost 24 times larger than the human body itself. The same is true for the other images with under-segmentation.

The last two images had a much higher accuracy (over 90%). The vast majority of segmentation with an accuracy of over 79% was over-segmentation errors. This can be seen in the example images – most of the human body is marked in yellow with small green areas. The most common mistake was leg over-segmentation. There might be several reasons why this was the case. First, some women who participated as actors in the data collection were wearing skirts or dresses. This type of clothing is not tightly wrapping the lower part of the body, and there is a steep depth gradient at the bottom of the clothing. It results in neighboring pixels having a large difference in the depth values, and the neighbor-based Euclidean clustering does not perceive it as the same object. This situation is shown in the fifth image – a woman is wearing a dress, and one of her legs was not marked as part of the body. The next issue is the *Kinect* sensor data capture quality with certain clothing materials. It was a common issue where the sensor does not capture the depth data for black denim, as expected

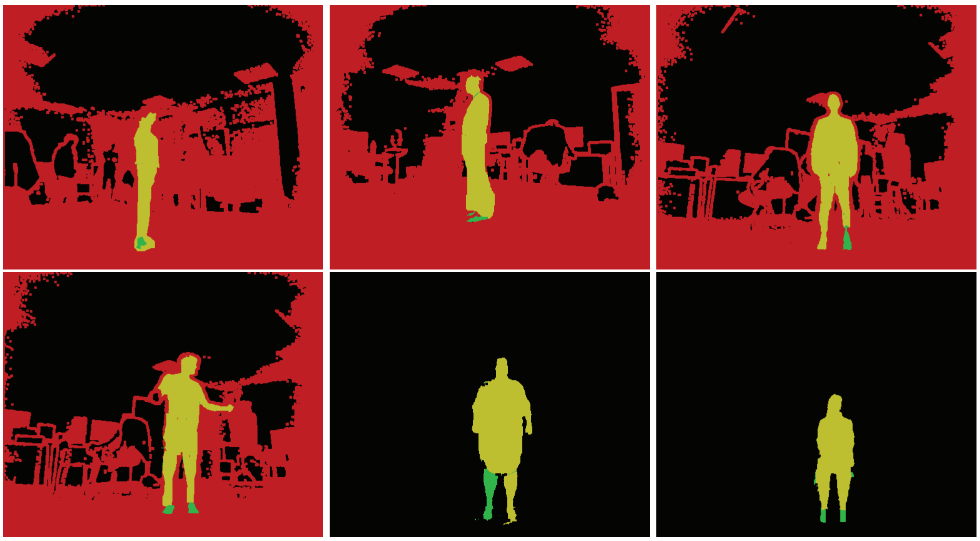


Figure 4.9. Euclidean-clustering-based segmentation error examples

from related research [80]. This material was usually worn as trousers, therefore, the leg segmentation quality was lower in those cases. Figure 4.3 shows such a case – the subject is wearing black jeans, and the no-filter image shows that the depth values are mostly zero for the legs. The device reports the depth values of zero, but they can still be segmented by selecting the zero-values blob in addition to the visible part of the body.

Figure 4.10 shows segmentation results for the bounding-box-based segmentation. The accuracies of the segmentation outputs are 4, 7, 11, 12, 16, and 96%, Qualitatively, the results are similar to the Euclidean-clustering-based segmentation results. The only difference is that under-segmentation is more aggressive and happens more often and to a greater extent. 4% accuracy is shown for both algorithms, however, the Euclidean-clustering-based segmentation had this accuracy with a smaller visible area of the human than the bounding-box-based segmentation. To reach the same accuracy with a larger ground truth point set, a larger segmented set is required. On the other hand, bounding-box-based segmentation had fewer problems with over-segmentation, which is also expected. The search volume is less fine-grained, therefore, it has the potential to capture more points than the Euclidean-clustering-based segmentation.

The opposite extrema are shown in Figure 4.11 for the Euclidean-clustering-based segmentation. These are the cases where the segmentation accuracy exceeded 98%. The best-recognized pose was the standing human, the front view, and there are only very minor differences between the ground truth and the output. These differences are due to the still present noise in the image as seen in the sitting position segmentation – there are several false-positive pixels on the ground around the feet. The chair was also segmented as a part of the object, however, this is expected since the algorithm has no semantic information on what type of object is expected, therefore, the human and the chair are considered a single object. This shows that these extrema are not

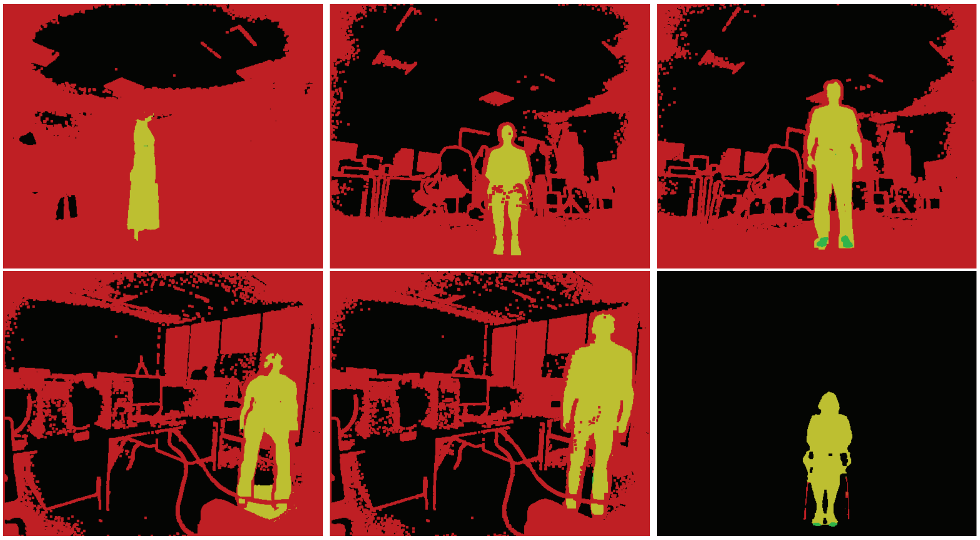


Figure 4.10. Bounding-box-based segmentation error examples

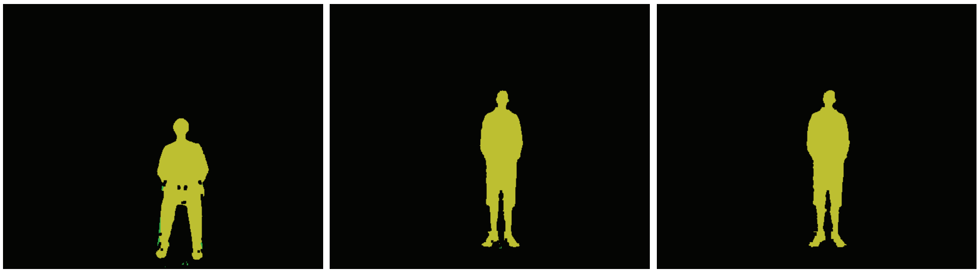


Figure 4.11. Euclidean-clustering-based segmentation over 98% accuracy frame examples

just anomalies in the segmentation output, but rather easily processed scenes for the Euclidean-clustering-based segmentation.

Similarly, high-accuracy frames for the bounding-box-based segmentation are shown in Figure 4.12. The output is semantically similar to the radius-search-based segmentation as there is some minor noise, and the chair is also a part of the object. Figure 4.8 shows that there is a large number of video sequences processed with the same accuracy as the last image in Figure 4.12, and semantically they are also processed with a high accuracy. The examples confirm that the highest-accuracy frames are not anomalies either for the proposed bounding-box-based algorithm.

To sum up, both algorithms have cases of under-segmentation. The bounding box has a higher tendency to under-segment the image, but the Euclidean-clustering-based segmentation is more likely to over-segment it. However, over-segmentation errors were smaller for both algorithms than under-segmentation errors.

Well-segmented frame and pixel percentage

Segmentation may be useful if a certain level of accuracy is reached. This level may be different for different cases of use, therefore, analysis with different accuracy

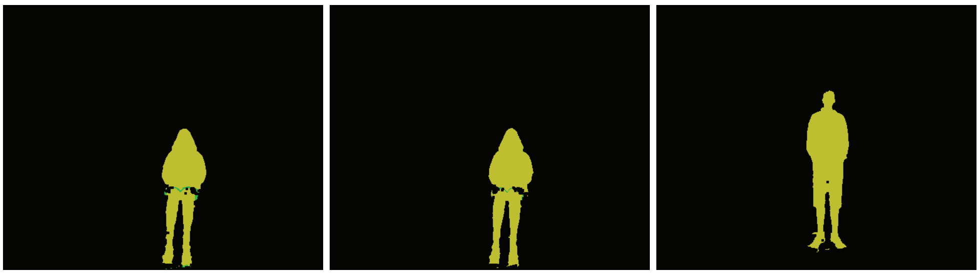


Figure 4.12. Bounding-box-based segmentation over 98% accuracy examples

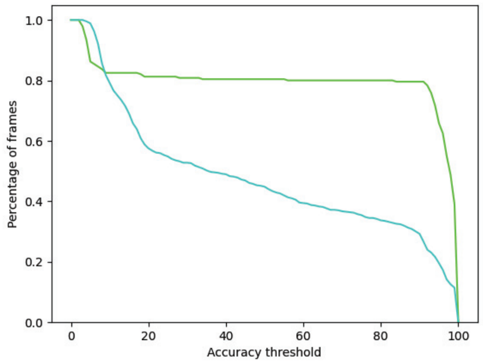


Figure 4.13. Euclidean-clustering-based segmentation threshold accuracy frame percentages

threshold values was performed. Frame percentages for both algorithms under certain thresholds are shown in Figures 4.13 and 4.14. The green curve represents the simple dataset, whereas the blue curve shows the complex dataset. The simple dataset frame count degrades similarly for both algorithms. There is a small percentage of completely incorrectly segmented frames at the 6-10% accuracy range. Then the curve descends very slightly up to around 90% accuracy, and then falls sharply. Both algorithms were capable of segmenting over 80% of the frames with 90% accuracy in the simple dataset. About half of the frames reached 99% accuracy. The graphs show that both algorithms can be applied to the simple dataset since they produce high accuracy for most of the frames.

The complex dataset is segmented with different accuracies, and it can be seen in the diagrams as well. None of the algorithms can be applied if high accuracy is required. 80% of the frames can be processed with around 15% accuracy for both algorithms. The Euclidean-clustering-based segmentation complex dataset curve descends more smoothly after the 20% mark. The difference between 20% and 90% accuracy is about 25% of the dataset for the Euclidean-clustering-based segmentation and only a few percent for the bounding box. While both algorithms cannot reliably process the dataset, the Euclidean-clustering-based segmentation offers better accuracy for a large fragment of the dataset, which may mean less human work to correct the mistakes.

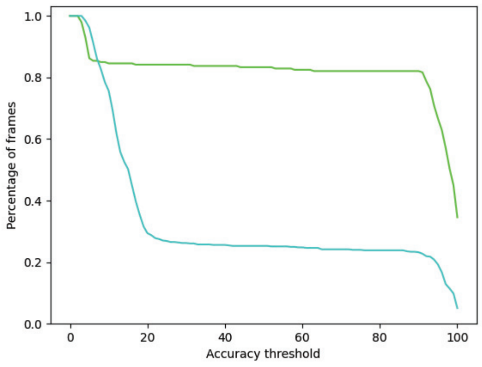


Figure 4.14. Bounding box threshold accuracy frame percentages

Table 4.5. Pixel-wise confusion matrix for Euclidean-clustering-based segmentation (simple dataset)

		Predicted		
		Background (negative)	Foreground (positive)	Total
Actual	Background (negative)	84.2%	12.4%	96.6%
	Foreground (positive)	0.1%	3.3%	3.4%
Total		84.3%	15.7%	100%

Tables 4.5 and 4.6 show pixel-wise confusion matrices for the adapted Euclidean-clustering-based segmentation algorithm for both datasets. False positives are much more prevalent than false negatives (false positive rates are 12.8% and 21.5% for simple and complex datasets, while the respective false negative rates are 2.9% and 4.6%). This is due to the under-segmentation problems already outlined in the previous sections.

Tables 4.7 and 4.8 show the confusion matrices for bounding-box-based search. The confusion matrices are computed pixel-wise. The class distribution between the foreground and background pixels is uneven as the size of the human in the frame is usually only 3-7% of the whole image area. The confusion matrices confirm that under-segmentation is much more prevalent as false-positives account for 11.4% of the pixel-wise classification output in the simple dataset (the false-positive rate is 11.8%), while false-negatives only account for 0.1% of the output (3% of the actual foreground pixels are misclassified). The false positive rate is higher with the complex dataset as 32.1% of the pixels are misclassified as the foreground (the false positive rate is 34.3%), but only about 3% of the actual foreground is misclassified again. These results further confirm that under-segmentation is the most common mistake made by

Table 4.6. Pixel-wise confusion matrix for Euclidean-clustering-based segmentation (complex dataset)

		Predicted		
		Background (negative)	Foreground (positive)	Total
Actual	Background (negative)	73.5%	20.1%	93.6%
	Foreground (positive)	0.3%	6.1%	6.4%
Total		73.8%	26.2%	100%

Table 4.7. Pixel-wise confusion matrix for bounding-box-based segmentation (simple dataset)

		Predicted		
		Background (negative)	Foreground (positive)	Total
Actual	Background (negative)	85.2%	11.4%	96.6%
	Foreground (positive)	0.1%	3.3%	3.4%
Total		85.3%	14.7%	100%

the segmentation solution.

Confusion matrices show that both algorithms tend to make similar types of mistakes as false negatives are very rarely observed. On the other hand, false positives are more common, especially with the complex dataset. The results suggest that both algorithms are comparable in accuracy on the simple dataset, with the bounding box approach having fewer false positives and false negatives, while the Euclidean-clustering-based segmentation approach is more accurate with the complex dataset in the same sense.

Accuracy analysis by the sex of the subject

The dataset used for accuracy evaluation consists of depth images with a balanced proportion of male and female participants. Body fat analysis in male and female bodies has shown that there are differences in the body fat distribution between the genders [345]. This finding suggests that body shapes may differ between males and females. Therefore, it may be useful to assert the differences in segmentation accuracies of the male and female participants.

The analysis has shown that female bodies are usually segmented with a slightly higher accuracy. The results are dissected by dataset, and the simple dataset is further dissected by pose. The Euclidean-clustering-based segmentation segmented females slightly better in the simple dataset in the standing pose. The accuracy difference for the sitting pose between males and females was lower than 0.05%. The complex dataset,

Table 4.8. Pixel-wise confusion matrix for bounding-box-based segmentation (complex dataset)

		Predicted		
		Background (negative)	Foreground (positive)	Total
Actual	Background (negative)	61.5%	32.1%	93.6%
	Foreground (positive)	0.2%	6.2%	6.4%
Total		61.7%	38.3%	100%

Table 4.9. Sex-wise accuracy of Euclidean-clustering-based algorithms

Algorithm	Simple – standing		Simple – sitting		Complex	
	Male	Female	Male	Female	Male	Female
Euclidean clustering	76.7%	**80.5%**	78.9%	78.9%	42.2%	**62.7%**
Bounding box	81.4%	**82.4%**	**81.6%**	79.6%	30.8%	**39.9%**

however, showed very different results, with the difference exceeding 20 percentage points in favor of females. The bounding-box-based segmentation shows similar results but is slightly shifted towards better male body segmentation accuracy. Simple dataset's standing poses and complex dataset are still segmented with better accuracy for females, however, the differences are about 2 times smaller. The simple dataset's sitting pose was segmented slightly more accurately for male bodies. The detailed results are outlined in Table 4.9.

These results may be determined by various reasons. Female clothing, such as skirts or dresses, do not represent the shape of the body, but rather has its own shape. This shape may be easier for the segmentation algorithms to capture as a single object. Next, the females participating in the research were shorter on average than the males. It was shown that both algorithms tend to under-segment the actual body. Since the body is smaller, there are fewer opportunities for the algorithms to diverge.

4.2.6. Total segmentation time cost analysis

The ultimate goal of semi-automatic segmentation is the reduction of the overall processing time (human time + software processing time). Therefore, both human time and processing time has to be estimated. Processing time analysis has been performed, and the results are presented in Section 4.2.4.1. Human time estimation, however, is a much more difficult process. Low-scale analysis is not viable because the data is highly variable, and different segmentation scenarios can be involved. Large-scale analysis is possible, however, it is less accurate because the person is sometimes distracted by their surroundings. Therefore, an approximate benchmark was selected.

The whole dataset has been segmented by two trained individuals. This was required anyway to get the training data for the Agrast-6 neural network. Exact measurements were not made, however, yet the time cost can be estimated by the file save timestamps. The times are accurate to the minute. It is also important to note that a slower version of the bounding-box-based algorithm was used at the time of this analysis (it took 16 ms per frame rather than the 6.3 ms achieved in further research).

The average total time spent on segmenting a depth video sequence was about 2 minutes. This involves loading the video sequence from the disk, running segmentation, human time, and other actions required to get a segmented image. This means that the total segmentation time was about 350 ms per frame. Previous analysis showed that the Euclidean-clustering-based algorithm can automatically segment about 80% of the simple dataset and about 25% of the complex dataset. If dataset sizes are taken into account, this is about 28% of the total amount of the data. This number is about 26% with the bounding-box-based segmentation following the same logic.

It was estimated that simply confirming that the segmentation is correct takes less than 1 second of human time. Since 12 frames can be seen at once on the screen, and they are processed in parallel, it takes 1 second of human time, 2.3 seconds of segmentation time with the Euclidean-clustering-based segmentation, and 0.016 seconds for the bounding-box-based segmentation. The bilateral filter takes 0.034 seconds in both cases. It also takes about 0.045 seconds to construct a search tree. Therefore, the full CPU processing time with the Euclidean-clustering-based segmentation is 2.4 seconds, whereas the bounding box takes 95 ms. The total time taken for all 12 frames is then 3.4 and 1.095 seconds, respectively. This means an average of 283 ms per frame with the Euclidean-clustering-based segmentation and 91 ms per frame with the bounding-box-based segmentation.

Since 26% of the data could be processed automatically, the processing time for the remaining 74% would have been 441 ms per frame to yield the 350 ms average. This means that an extra 350 ms per frame is required if the auto segmentation does not work immediately. The effect of replacing the bounding box with the Euclidean-clustering-based segmentation is difficult to estimate precisely, so some assumptions will have to be made. First, a slightly larger proportion of the data was processed accurately, and this has to be taken into account. Secondly, there are many cases where the Euclidean-clustering-based segmentation provides a more accurate result, which would result in a lower amount of manual work to correct it. Let us assume that this would result in 30% less manual work for the inaccurately processed frames. High-accuracy frames would require 1 second of manual work + 2.4 seconds of processing time per 12 frames, so 283 ms per frame for 28% of the data. Low accuracy frames would require 432 ms per frame (346 ms of manual work x 0.67 x 12 frames x 2.4 seconds CPU time is 5.2 seconds per 12 frames) for 72% of the data. This leads to an average of 390 ms per frame, which is 40 ms higher than the bounding box approach. Given the size of the dataset (262k frames), this would save almost 3 hours of total work required to segment the whole dataset.

It should be noted that marking each frame separately is many orders of magnitude slower as it was estimated to take about 20 seconds to mark it fully manually. Even such techniques as marking one frame fully manually and then transferring the manual segmentation to the subsequent frames substantially helped in time reduction as the segmentation only needed minor corrections. The overall time savings for a 262k frames dataset with a time reduction from 20 to 0.35 seconds is almost 60 days of non-stop work savings (1455 hours vs. 25 hours of work). This allowed a team of only two people to segment the whole dataset in a matter of days. If segmentation was a paid activity, given the minimum salary in Lithuania of 4.47€ per hour as of 1 January 2022, it would have saved about 6400€ (6500€ vs 114€).

It should also be noted that the further optimized version of the bounding-box-based segmentation would save an extra 9 ms of time per 12 frames, which is a minor improvement where the total time is 4.2 seconds per 12 frames, however, this would have saved about 3 minutes of the processing time for the whole dataset.

4.3. Under-segmentation correction experimental evaluation of the recursive 2-Means split algorithm with a random forest classifier for split acceptance

4.3.1. Experimental evaluation of the performance

The same benchmark was used for experimental performance evaluation as described in Section 4.2.4.1 – that is, 1000 frames have been fully segmented by using the suggested correcting algorithm. The results are compared with the best-performing improvements described in the previous sections. The same improvments are also used as the base algorithm, hence, all performance overhead is purely due to the correcting algorithm.

The correcting algorithm is guaranteed to run slower because it is the standard segmentation + the added corrections, i.e., it is a superset of the actions performed by the standard segmentation. Node traversals are also evaluated, but they are also going to be higher. The correction fights under-segmentation by returning part of the segmentation output to the search tree. This means that the base segmentation will be repeated more times and may traverse the same node multiple times by marking it as removed, then marking it as non-removed, and marking it as removed again. However, the scale of the extra work has to be measured via the benchmark.

Table 4.10 shows the benchmark results and their comparison to the global baseline of this research (Euclidean-clustering-based segmentation) and the fully optimized version of the expanding bounding box proposal. The node traversal count increase is small at around 10% compared to the base bounding-box-based algorithm. The runtime increase, however, is a lot larger. ~40.6 ms per frame is the computation cost of the proposed corrections compared to 6.33 ms of the total segmentation time required for segmentation without corrections. It is important to note that the output of the algorithms is also different. However, it is still almost 50 times faster than the standard radius-search-based segmentation.

The dissected performance by the type of segmentation is shown in Table 4.11. Most of the time is taken by marking points as non-removed in the search tree. This was the only activity with the computational complexity $O(k \log n)$, therefore, it is the slowest part of the algorithm. Segmentation is taking a bit longer (over 7 ms) than the base algorithm because there is more segmentation to do, which is also reflected in the

Table 4.10. Algorithm performance comparison

Algorithm	Runtime, ms	Node traversals, M
Euclidean clustering	2326	194
Bounding box	**6.33**	**0.311**
Bounding box + correction	46.9	0.342

Table 4.11. Computation time dissection by activity

Marking points as non-removed	13.9 ms
Metrics	8.81 ms
Segmentation	7.07 ms
K-Means partitioning	6.23 ms
Prediction	4.14 ms
Centroid computation	3.44 ms
Removed point collection	482 µs

node traversal comparison. Metrics computation, K-Means partitioning, and centroid computation take a similar amount of time, which reflects their linear complexity.

4.3.2. Experimental evaluation of accuracy

4.3.2.1. Overview

The benchmark has shown that the simple dataset saw a slightly negative impact in accuracy when the corrections were applied. This dataset was processed with a high accuracy without the corrections, and under-segmentation was rare. However, classifier recall values are not 100% and some incorrect cuts are accepted. The effects of the imperfect recall and under-segmentation reduction seem to have canceled each other out, and the overall average accuracy is similar with and without the correction. The difference is 0.4 percentage points. The complex dataset was processed more accurately, and the improvement is noticeable. This effect is expected because the correction is meant to solve the problem of under-segmentation which is the most apparent in this dataset. Using the correction raises the accuracy of the bounding-box-based search higher than the level of the Euclidean-clustering-based segmentation at a performance penalty. The added accuracy is huge, and it is 21.9 percentage points and 7.2 percentage points higher than the Euclidean-clustering-based segmentation. The summary of the average accuracy values is provided in Table 4.12.

Table 4.12. Average accuracy comparison

Algorithm	Simple dataset accuracy	Complex dataset accuracy
Euclidean clustering	79.8%	48.1%
Bounding box	82.1%	33.4%
Bounding box + correction	82.5%	55.3%

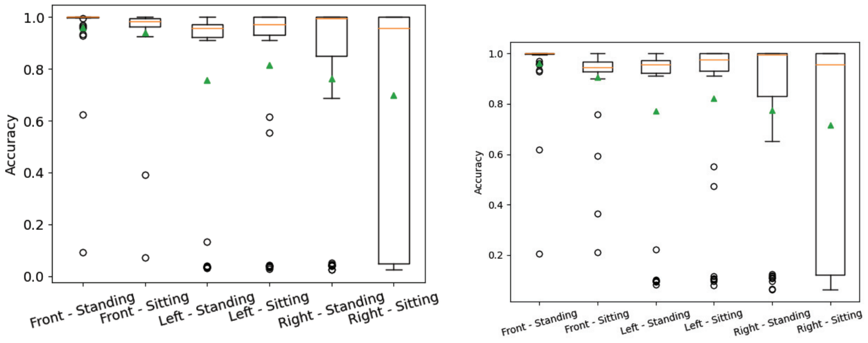


Figure 4.15. Simple dataset accuracy comparison. Left – without corrections, right – with corrections

4.3.2.2. Detailed result analysis

A simple dataset was processed with very similar accuracy with and without corrections. The right sitting pose had very similar average accuracy, but the Q1 value got slightly higher with the corrections. The front sitting pose was, however, processed with a lower accuracy with the corrections applied. There are 4 low accuracy outliers instead of 2, and Q1, Q2 and Q3 values are also all lower with the corrections. In general, the results for the simple dataset reflect the recall values observed during random forest training – i.e., that a small proportion of the samples is marked incorrectly, but low accuracy frames are generally improved. Since this dataset had very few samples with under-segmentation (which is visible in the box plot in Figure 4.15), the added overall accuracy is probably not worth the extra computation time.

The complex dataset, however, saw a decent improvement in accuracy. The back view had a similar accuracy average, but the Q1 value improved. The front and side views were processed with a much higher accuracy. The average front accuracy jumped from 21% to 59%, and the side accuracy – from 9% to 34%. The front segmentation accuracy was mostly in the 22-90% accuracy range, whereas, without the improvement, everything above 25% was considered outliers. The side view did not see such a spike in accuracy, however, it consistently improved, and the average of 34% is higher than the best accuracy across the whole dataset without the corrections. Overall, the accuracy is even better than by using the Euclidean-clustering-based segmentation. Generally, the cuts provided by the correcting algorithm show improvements in accuracy in under-segmentation scenarios. However, the stability of the segmentation accuracy was not achieved as the accuracy values still range from very low to very high (except for the side view where the maximum accuracy is only about 60%). The average values are, nevertheless, largely improved, which shows that the corrections do not guarantee the correct output, but still increase the likelihood of a better segmentation accuracy. All details are outlined in Figure 4.16.

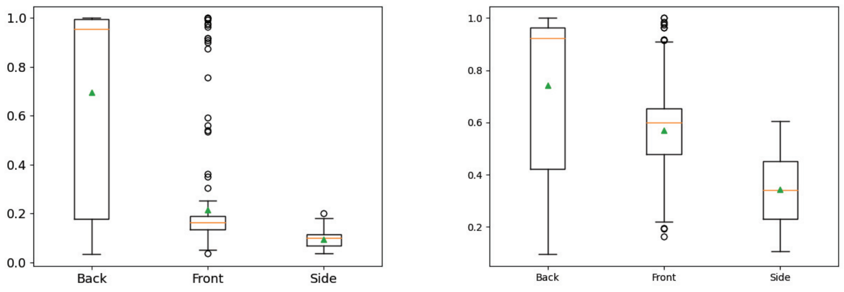


Figure 4.16. Complex dataset accuracy comparison. Left – without corrections, right – with corrections

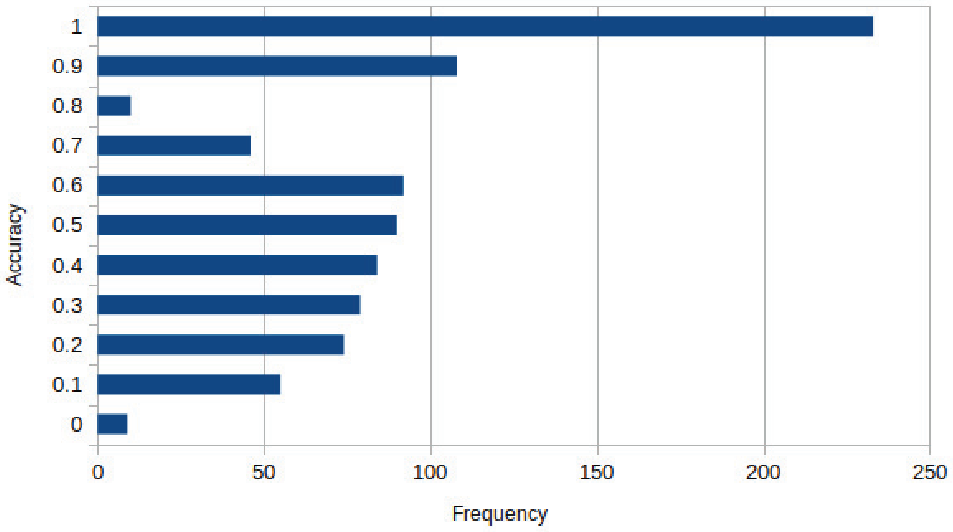


Figure 4.17. Corrected segmentation accuracy histogram

4.3.2.3. Accuracy histogram analysis

Accuracy histograms presented for the bounding-box-based segmentation and the Euclidean-clustering-based segmentation had similar shapes – spikes in very high and low accuracy and dips in the mid-accuracy range (see Figure 4.8). The histogram is different when the correction is added. There are fewer samples in the above-90% accuracy range as part of them moved to the 80-90% range. However, the low-accuracy segmentation no longer has a spike as this spike was spread through the mid-accuracy range. Even more, the number of samples continually increases going from the sub-10% range to the 50-60% range. One of the conclusions about the bounding-box-based segmentation histogram was that this algorithm is unstable – it either yields a very good accuracy, or a very low accuracy. The accuracy histogram of the random-forest-based algorithm is presented in Figure 4.17.

These results show that the corrections of under-segmentation fix the worst cases of under-segmentation and improve the accuracy a lot. Accuracy, as before, is measured by using cross-set intersection metric. Going from 21% accuracy to 59% means that the under-segmentation is reduced from 3.7 times larger than the human body size to 70% of the human body size. That cuts the under-segmented area 3 times larger than the object that had to be segmented. Similarly, going from 9% to 34% means the under-segmented area reduction from 10.1 times to 1.9 times the area of the actual object. These improvements are major in the context of semi-automatic segmentation. This suggests that the area the human would have to manually 'erase' is reduced by a significant amount, which saves extensive manual work.

4.3.2.4. Qualitative error analysis

Figure 4.18 shows some examples of the mistakes left after the corrections have been applied. The images reflect 10%, 11%, 44%, 46%, 47%, and 96% accuracy values. The two initial frames show the same type of error – part of the under-segmented output was cut off (from all sides, multiple cuts were required), however, part of it was left because the human body left in a 'bubble' of an under-segmented area. There is no way to split the segmentation output further based on 2-Means so that the human body would stay intact. The next three examples show that most of the under-segmented area was cut, however, a small piece of it was left. The accuracy is sub-50% because some degree of over-segmentation also occurred, and the score got lower. The proposed correction algorithm does not aim to solve the over-segmentation errors. The presently discussed images could have their accuracy further improved by applying a post-processing step because part of the under-segmented output is left as a separate object. However, this would further increase the runtime. The final image required no corrections in terms of under-segmentation, and those corrections were not applied. These errors are also easy to fix manually by a human as the mouse does not have to be controlled carefully since the erasing is applied far from the human body.

The error examples suggest that the error reduction is significant in the sense of the manual work required to fix the segmentation manually. There was a very small amount of full-background under-segmentation as the cases shown in Figures 4.10 and 4.9. The low-accuracy segmentation is still of much better quality than the low-accuracy segmentation without the correction which both had lower cross-intersection scores and were much more prevalent without the corrections.

The best-segmented frames are the frames where the bounding box segmentation was correct, and this output remained unchanged. Therefore, most of these images are very similar to those shown in Figure 4.12.

4.3.2.5. Well-segmented frame percentage

Analysis of accepted frames by threshold accuracy was performed similarly to those shown in Figures 4.13 and 4.14. Again, the green curve shows the results for the simple dataset, whereas the blue curve represents the complex dataset in Figure 4.19.

The simple dataset was segmented with a similar accuracy with and without corrections, therefore, the curve has the same shape with two steep dips at around 6-10%

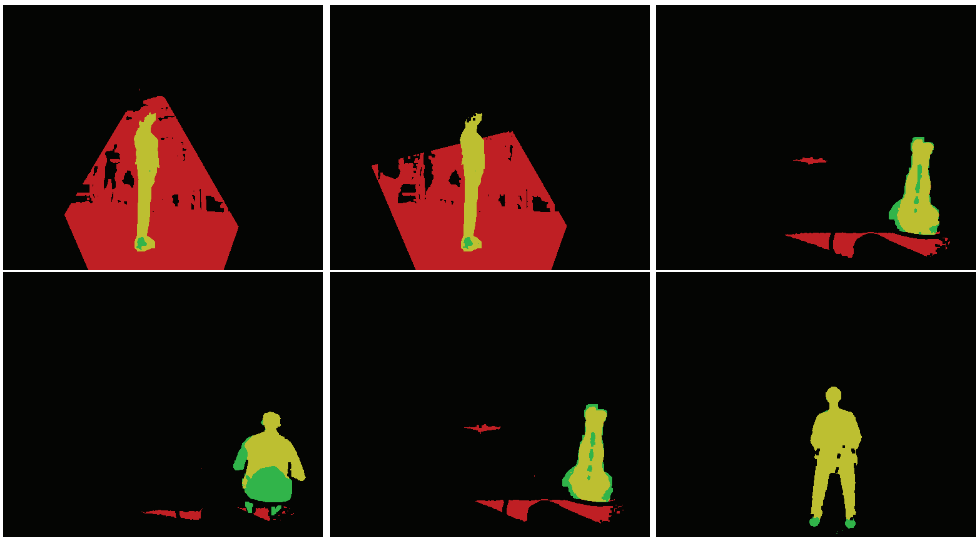


Figure 4.18. Recursive 2-Means split algorithm with a random forest classifier for split acceptance correction error examples

range and at around 90% mark. The complex dataset, however, has a very different graph shape. It falls smoothly from around 20% to around 70% accuracy values, and then falls again after 90%. This shows that if an accuracy of 90% is required, the correction does not solve the problem – the amount of usable segmentation is around 23% in both cases. However, if lower values are acceptable, the random-forest-based corrections change the fitting frame proportion greatly. The number of frames segmented with 20% accuracy or better increases from around 30% to over 95%. Similarly, the 50% accuracy threshold contains about 25% of segmentation outputs without corrections and about 50% of segmentation outputs with corrections. Therefore, the corrections improved the 'good enough' frames proportion by a large margin if the imperfect segmentation output is acceptable.

Table 4.13 shows the pixel-wise confusion matrix for a simple dataset. This table can be compared to Table 4.7, which shows the same matrix for the algorithm without the recursive 2-Means split algorithm with a random forest classifier for split acceptance corrections, so it can be considered as a baseline. The true positive percentage is at the same level as for the baseline algorithm as well as the false negative percentage, which is 97% of all positive pixels are found correctly in both cases. The background pixels, on the other hand, are classified more correctly when the corrections are applied. True negatives account for 92.5% of all pixels vs. 82.5%, while false negatives, in turn, account for 4.1% vs. 11.4%, which shows that the false negative rate is greatly reduced (from 11.7% to 4.2%). Since false positives indicate the problem of under-segmentation, this further confirms that the algorithm does what it is expected to do, i.e., it reduces under-segmentation (false positives) while keeping the accurate segmentation output unchanged. Since the false positive rate is reduced, fewer manual corrections are required. It could save some user time at a cost of slightly reduced

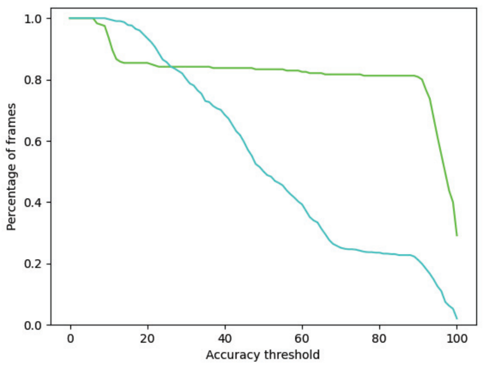


Figure 4.19. Recursive 2-Means split algorithm with a random forest classifier for split acceptance correction accuracy frame percentages

Table 4.13. Pixel-wise confusion matrix for random-forest-corrected segmentation (simple dataset)

		Predicted		
		Background (negative)	Foreground (positive)	Total
Actual	Background (negative)	92.5%	4.1%	96.6%
	Foreground (positive)	0.1%	3.3%	3.4%
	Total	84.3%	15.7%	100%

performance.

Similarly, Table 4.14 shows the pixel-wise confusion matrix for a complex dataset. It can be compared with Table 4.8 which can be treated as a baseline. The recursive 2-Means split algorithm with a random forest classifier for split acceptance corrections introduces some trade-offs for this dataset. First, the true positive rate is reduced from 96.9% to 87.5%. This indicates that the classifier rejects some true positive pixels as negatives. However, the true negative rate is greatly improved from 65.7% to 94.8%. This improvement is an indicator of the under-segmentation reduction by a large margin. It is very important in the light of semi-automatic segmentation – the amount of manual corrections is greatly reduced. A reduced rate of the true positives is a smaller issue because it is easy to add segments by using the proposed methods – segmentation can be repeated by using automatic algorithms. Contrary to that, the excessively large segmentation output has to be corrected manually. Therefore, this contributes a lot to the reduced total segmentation time. Combined with the fact that the true positive rate is almost unaffected for the simple dataset, applying the corrections could result in a large amount of manual work savings without adding substantial manual work for simple data.

Table 4.14. Pixel-wise confusion matrix for random-forest-corrected segmentation (complex dataset)

		Predicted		
		Background (negative)	Foreground (positive)	Total
Actual	Background (negative)	88.7%	4.9%	93.6%
	Foreground (positive)	0.8%	5.6%	6.4%
	Total	89.5%	10.5%	100%

Table 4.15. Sex-wise accuracy of corrected segmentation

Simple – standing		Simple – sitting		Complex	
Male	Female	Male	Female	Male	Female
82.3%	**83.5%**	**81.0%**	80.2%	52.5%	**62.2%**

4.3.2.6. Accuracy analysis by the sex of the subject

Accuracy by sex analysis shows that the corrections do not depend on the sex in any way. The male and female accuracy was similar with or without the corrections for the simple dataset. Males were segmented with a slightly better accuracy in the sitting poses, while the females were in the standing poses. The complex dataset also shows a better accuracy for females with a similar difference of 9.7 percentage points (vs. 9.1 without the correction). Detailed results are provided in Table 4.15.

These results were expected since the corrections were based on the metrics that were not intended to be related to the sex of the subject. In addition to this, there was a consistent improvement in the average accuracy across both genders – all the six measured dissections saw at least some improvement. This confirms improvement invariance to the sex of the subject.

4.3.3. Impact on the total segmentation time cost

Since the suggested accuracy improvements raised the accuracy of the segmentation and extended the processing time, the total effect should be estimated. First, the added processing time is about 40 ms. Since 12 frames are processed in parallel and the amount of data is 262k frames, this would add almost 15 minutes of the processing time. Next, some time would be saved due to improved accuracy as fewer manual corrections are required if the original accuracy is better. The corrective algorithm showed the most improvement for the complex dataset where the average accuracy went from 33.4% to 55.3%. This means that the area that had to be removed manually went from 2x human body size to 0.8x human body size on average. It was estimated that the added time cost for a badly segmented frame was 350 ms. The exact time savings are difficult to estimate, however, the under-segmented area was reduced by a factor of

2.46. The time savings would be smaller since the human has to evaluate the segmentation quality (about 1 s per 12 frames), select the erasing tool, and make other mouse movements that are not directly related to the degree of segmentation incorrectness. A rough estimate could be that 2.5 s of out 4.2 s per 12 frames is the actual work of removing under-segmentation. This means savings of ~140 ms per frame. The amount of poorly segmented frames was 74% of the dataset, which means that 0.14 s is saved for 194k frames each, thus totaling at the time savings of ~7.5 hours. Even though this analysis is rough, the added processing time is 30 times shorter than the estimated manual time savings. Even if the estimate is incorrect by an order of magnitude, applying the corrections would still result in total time savings.

This analysis shows that the added computation complexity is a small price to pay in comparison to the increased accuracy. However, in order to achieve even better results, the bounding-box-based algorithm without corrections could be used for the simple dataset since the accuracy gain is very small for it, and the algorithm with the suggested corrections could be used for the complex dataset where the under-segmentation reduction is significant.

4.4. Agrast-6 training and evaluation

4.4.1. Agrast-6 training process

A binary cross-entropy loss function was used during Agrast-6 training. It is a common choice for training binary classification models, as it measures the distance between the predicted probability distribution and the true distribution. A significant decrease in both the training and validation loss suggests that the model is learning effectively, and that it accurately predicts the probabilities of the positive class.

One potential reason for the decrease in loss is that the model has learned to make more accurate predictions, which results in a lower distance between the predicted and the true probability distributions. Another possibility is that the model has learned to better fit the training data, which can also lead to a decrease in loss.

Overall, the decrease in loss is a good indication that the model is learning effectively and making accurate predictions. The binary cross-entropy loss function is well-suited for this task and has likely contributed to the model's success in improving its predictions over the course of training.

It appears that the Agrast-6 machine learning model has made significant progress during training. The training loss decreased significantly, from 0.077 to 0.021, which suggests that the model is learning effectively and improving in its ability to make predictions on the training data. Similarly, the validation loss decreased from 0.09 to 0.036, which indicates that the model is also generalizing well on the new, unseen data.

The training precision, binary accuracy, and recall all increased, which indicates that the model is becoming more accurate in its predictions on the training set. For example, the training precision improved from 0.859 to 0.941, which means that the model is correctly identifying a larger proportion of the positive examples as positive. Similarly, the binary accuracy increased from 0.972 to 0.991, which suggests that the model is making fewer incorrect predictions overall. The increase in recall from 0.454 to 0.897 suggests that the model is becoming better at identifying all the positive examples in the training set.

In addition to these improvements on the training set, the model also showed strong generalization capabilities, as evidenced by the improvements on the validation set. The validation precision, the binary accuracy, and the recall all increased, which indicates that the model is making more accurate predictions on the validation set. The AUC (area under the curve) also increased on both the training and the validation sets, which is a measure of the model's ability to distinguish between the positive and negative examples. A higher AUC indicates that the model is doing a better job at this task, and the increase from 0.970 to 0.997 on the training set and from 0.965 to 0.990 on the validation set suggests that the model is becoming increasingly effective at this task.

Overall, these results are encouraging and suggest that the model has made significant progress during the first 185000 training steps (about 1.5 epochs). The improvements on both the training and validation sets, as well as the high AUC scores, suggest that the model is performing well and may be ready for further evaluation or deployment. The training progress is shown in Figure 4.20.

The training progress of Agrast-6 as of epoch number 4 is visualized in Figure 4.21. The training metrics (the loss, binary accuracy, precision, and recall) did not seem to improve from epoch 4 to 9 a lot as they oscillate in the same interval. However, the test metrics slightly improved. The loss decreased from 0.043 to 0.0325, and the recall increased from 90.3% to 91.9%. The precision and binary accuracy did not change as much.

These results suggest that the neural network is still learning after 9 epochs. The results also show that the model does not suffer from overfitting since the metrics for the test and the train datasets are similar. The training metrics are slightly higher, so there is a small amount of bias as the model works slightly better on the data it has already seen. The differences are small, with the largest difference being the loss function value (0.0162 for the training data vs. 0.0325 for the test data), however, the loss for the test data consistently decreased with each epoch with one exception. The fact that the test loss is decreasing while the training loss is not might indicate that there are many local minima in the loss function, and the optimizer is picking a different one each time, however, the test loss is decreasing, so the model seems to generalize quite well.

4.4.2. Qualitative analysis of the training progress

An insight into how the network learns the features of a human body can be seen in Figure 4.22. The charts show the early progress of training. A test depth image from the test dataset was selected to monitor the training process, and it is shown in Figure 4.23. The first image shows the output of the neural network after merely one batch of data (4 images). Since the training started with random weights, the whole depth scene is just randomly distorted, and not much is visible in the output. The second image shows the output after 100 training batches. The human shape in the center is already visible, however, the network confidence is not high yet (the intensity of the

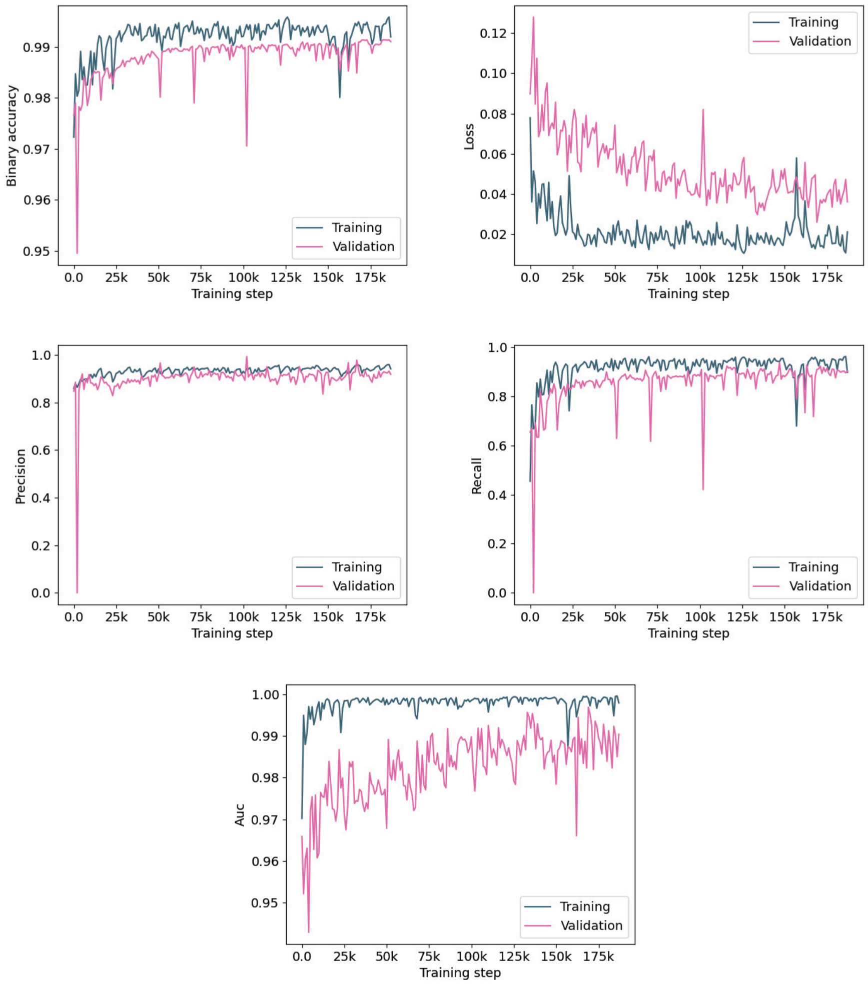


Figure 4.20. Agrast-6 training progress data

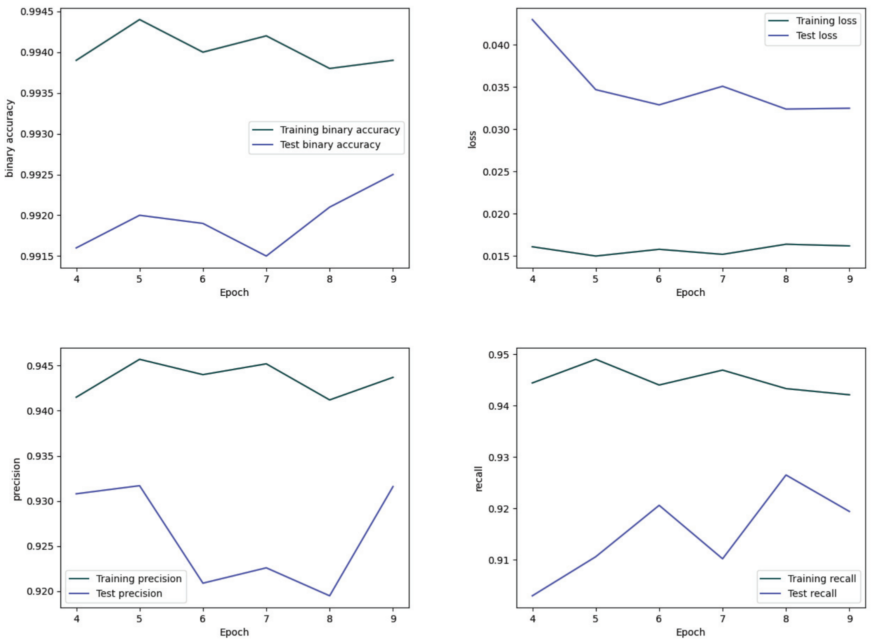


Figure 4.21. Agrast-6 training progress data

white color reflects the confidence). Some artifacts around the silhouette also exist, yet some of them are other objects in the scene. The third image shows how much the network learned in 8400 batches (33.6k images). The human shape is clearly visible, and the confidence of the network is much higher, however, the areas around the legs and the head are what the network is not sure about. The final image is the output after 16.2k batches (64.8k images, near the middle of the first epoch). The confidence of the network is higher again across the whole human body, however, the area around the head is still not correctly recognized.

The output of the network after epochs 2-9 for the same test depth image is shown in Figure 4.24. The initial images indicate that the network struggles with the head area the most. It is blurry, and the confidence is lower than the rest of the body. However, it increases with each epoch, and the final image looks best qualitatively. This test image introduces another challenge – the shoes of the person in the test frame are black, and *Kinect* could not capture it properly. The shoes were marked correctly in the ground truth for such cases, however, it is difficult for the network to infer the shoes from non-existing data. On the other hand, the rest of the body is segmented correctly and with higher confidence than in the previous epochs. These results correlate with the loss function decreasing with each epoch. This confirms that the selected loss function fits the problem being solved, and that the training process correctly optimizes it. However, the head is a relatively small part of the body, and therefore it might contribute less to the loss function.

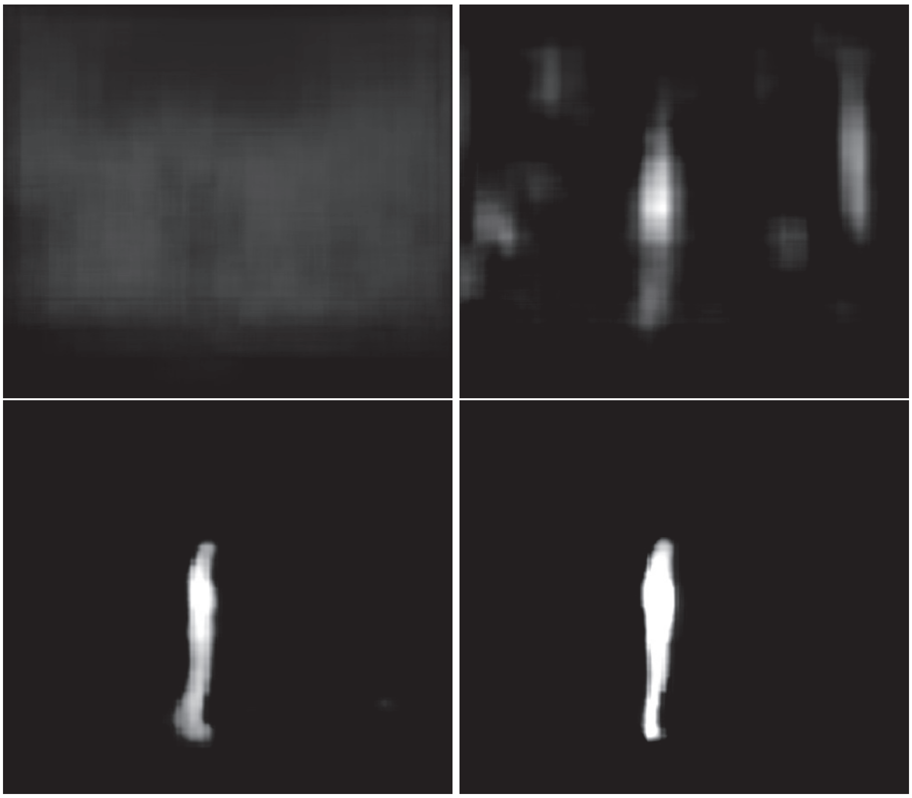


Figure 4.22. Neural network output after training batches 0, 100, 8400 and 16200

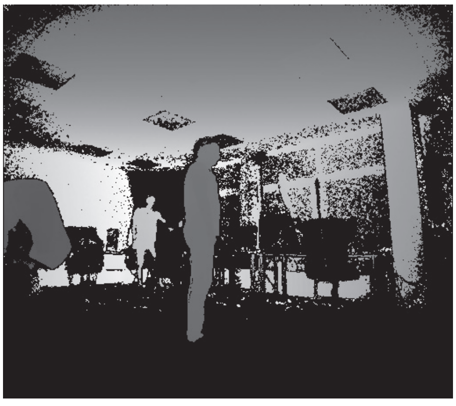


Figure 4.23. Original depth frame used for qualitative testing

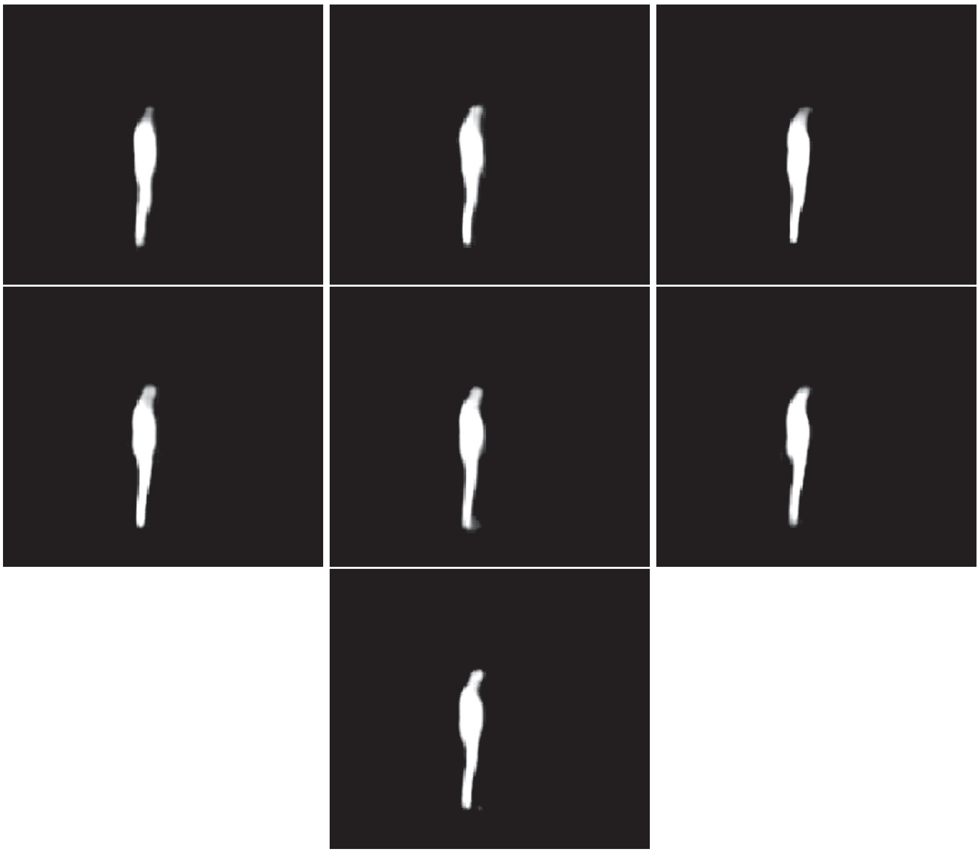


Figure 4.24. Neural network output after epochs 2-9

4.4.3. Image processing analysis

Appendix B contains images showing the output of each layer in the Agrast-6 network. Layer 0 is the input. Layer 1, the first convolutional layer, generates 32 images from the input. Most of the images are similar to the original with varying depth intensities. This is expected as the convolutional filter window size is small, and the image is comparatively large. The second layer is max-pooling, so the images are processed as expected (the local features are contracted as only the maximum value passes the filter). Since the max-pooling operation does not change the input image as much, layers 3, 5, 7, and 9 are skipped. Layer 4 is another max-pooling layer with a convolution applied in layer 3 which produces 128 images. However, most of the images in this layer's output are dark, which means that there are few activations. Only 4 images are visually distinct from the others. This might be an indication that the network does not utilize all the available depth to learn the features. Layer 6 (the final max-pooling layer with another convolution applied) is the most interesting as it is the final layer in the encoder and represents how the encoder encoded the features. It produces 256 images, and most of them have a visually distinct vertically oriented shape in the place where the human was standing. The resolution of these images is low (14×16 pixels), but they seem to encode the standing human position. There are 2 images with larger activations. However, there is a sizeable amount of near-empty images, which, again, suggests that – maybe – 256 images for feature encoding is too much.

The following images show how the decoder reconstructs the image. The first upsampling layer (layer 7) is not shown as the output is similar to layer 6. Layer 8 (the first transposed convolutional layer) has a much more interesting output. Human shapes in different forms are visible in the images as well as some artifacts. There are also images where the human shape is not activated, but the 'halo' around the body is decoded, which shows that the network learns about the boundaries of the silhouette in multiple ways. Layer 9 is the second upsampling layer, and it has a very similar output to layer 8, so it is also not shown in Appendix B. The output of layer 10 is much closer to the actual expected output where human silhouettes (sometimes inverted) are visible in different forms. Some images contain something similar to the expected final output of the whole network. Most of the images also contain an artifact on the left (the back of a chair). It is worth noting that, while most images agree on the basic shape of the human, their interpretation of the actual boundaries is different. Sometimes the silhouette is too thin, sometimes too thick. However, this part of the decoder shows that most of the images carry some information about the human body and probably contribute to the final output. Layer 11 is an upsampling layer, however, its output is also present in the appendix since it is the final layer before making the final prediction and because it is interesting to analyze how it upsampled the images. The resolution of the images went up four-fold, and the edges of the silhouettes are therefore somewhat little blurry. This seems to be the main source of accuracy loss in the network – here is the main trade-off between the performance and accuracy. The network tries to compensate for this from the previous layer where the human silhouettes have different thicknesses.

The aggressiveness of upsampling most probably causes different interpretations of the silhouette in the previous layer so that the network can reconstruct the silhouette in the upsampled image. This may also be the problem why the head was so difficult for the network to learn – the head in the downsampled image becomes very small, and therefore it is difficult to upsample it correctly even when having many representations. These representations are recombined in layer 12 (the last convolutional layer of the decoder). This layer has multiple versions of the human body without any other artifacts, some versions with artifacts, and some empty images. This, again, suggests that a depth of 32 might still be too much since there is no information stored in some images. One more interesting note is that there are multiple images in this layer where the human body has feet, however, the network decided to omit them in layer 13 (the final convolutional layer). There are two inverted human silhouettes without feet, and it seems that they contributed the most towards this final representation.

Overall, the network seems to process the test image reasonably, as it encodes the features that are then gradually decoded into the human silhouette with the background removed in multiple different ways. However, some images in some layers seem to contain no information related to segmentation, and they might be unnecessary to produce the correct output. Nevertheless, this architecture only has 1.25M parameters and seems to process images both correctly and reasonably, which confirms that this simpler architecture is viable and also learns the human body interpretations.

4.4.4. Performance benchmark

The benchmark has been run by using the same 1000 test frames as in the previous benchmarks. The average prediction time was **166 ms**. The time was relatively stable – the fastest prediction was made in 34 ms, which is very close to the average. The standard deviation was only 12.8. The only outlier was the first frame which took 229 ms. This may be a result of just-in-time compilation in Java where the code is compiled as required, which happens as it is run for the first time. Repeating the same benchmark directly in the Python code reduced the prediction time to **34 ms** per frame.

This performance number could be put into perspective with 442.5 ms achieved by the *SegNet* neural network. The researchers used NVidia Titan GPU which is now superseded by modern GPUs as well as a now-outdated Caffe library version. NVidia 1660 Super used for this research is a more powerful GPU. However, the images being processed are 25% larger for this research. Even with additional required data transformations, Agrast-6 shows a much better performance for automatic segmentation. The model size on the disk is 15.4 MB, which is a great reduction from the 117 MB model size of *SegNet*. Further investigation is provided in Section 4.4.5.7.

4.4.5. Accuracy analysis

The same analysis of accuracy has been performed as for the manual segmentation methods. The ground truth data is also re-used – it has been acquired by using the full semi-automatic segmentation solution described in Appendix C.1.

Table 4.16. Average accuracy per dataset

Simple dataset	82.1%
Complex dataset	88.6%

4.4.5.1. Overview

The average cross-set intersection was measured for simple and complex datasets separately. It was computed by using the ground truth and predicted binary masks. The achieved results were 82.1% for the simple dataset and 88.6% for the complex dataset. The results are also shown in Table 4.16.

The simple dataset contained about 3 times fewer data, therefore, this imbalance in accuracies may be caused due to the imbalance in dataset sizes. This may also mean that 9 epochs of training are not enough and that the training process should still be continued. On the other hand, the accuracy achieved with the complex dataset is very high, much higher than whenever using semi-automatic segmentation methods. This shows that Agrast-6 was able to learn even the complex structures given a large amount of data to learn from. However, the accuracy achieved with the simple dataset is on a similar level as when using the semi-automatic methods.

4.4.5.2. Detailed result analysis

Simple dataset processing accuracy depends on the pose of the human. Standing poses were processed with a better accuracy from all angles. The camera angle also had some impact on the accuracy. The front view had the best accuracy, while it decreased for other views. It looks like the silhouette size had some impact on the accuracy. The front camera saw the largest surface area of the human body. This issue was already presented previously when analyzing intermediate network activations. The edges of the object are blurry in the decoder layers, and this gives a higher penalty when the total surface area of the silhouette is smaller.

The number of low-accuracy outliers is small, while Q1 accuracy values are quite high compared to the proposed bounding box method. Since the average accuracies for the simple dataset are similar, Agrast-6 is more stable, however, it does not achieve extremely high accuracies. The bounding box method is less stable – it either segments the human body correctly or near-correctly, or fails completely.

The visible surface area of the silhouette had an impact when segmenting the complex dataset as well. The back side was segmented with the highest accuracy, while the side view was more difficult to segment. The boxes in the box plots in Figure 4.25 are narrow; most frames fall into a narrow range of accuracy values for each camera view. This again shows the segmentation stability. Q1 values for each side exceed 90%, and the overall average accuracy only falls below 90% due to some lower-accuracy outliers.

The observed results suggest that the model has learned the features of the human silhouette, however, more data might be needed for the simple dataset to achieve a better accuracy. On the other hand, the complex dataset is processed with a better

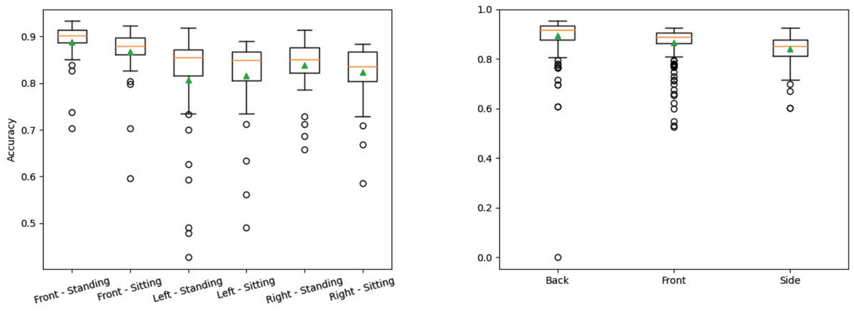


Figure 4.25. Accuracy analysis for simple and complex datasets for Agrast-6

accuracy even though the data in the complex dataset is more diverse – there are only 2 different poses in the simple dataset and 30 poses in the complex dataset. However, more data leads to a better accuracy, even though the problem seems more difficult at a first glance.

4.4.5.3. Accuracy histogram analysis

Figure 4.26 reveals the high percentage of high-accuracy frames when using Agrast-6. The bars in the histogram below 30% accuracy are invisible – there was only 1 video sequence in the 10-20% range, and other ranges are empty. The sub-60% range is also nearly empty. Most of the frames (over 92%) fall into the 80-100% accuracy range (as a reminder, a 0.9 value in the histogram means the accuracy interval $[0.9, 1)$). As another reminder, 80% cross-set intersection accuracy may mean one of two things. In the case of over-segmentation, at least 80% of the actual human shape is captured, and in the case of under-segmentation, the under-segmented area is at most 25% of the size of the human silhouette (if the human silhouette size is referred to as 1, the total segmented area must be 1.25 as $\frac{1}{1.25} = 0.8$). If both over-segmentation and under-segmentation occur, their effects are multiplied, hence the 'at most' and 'at least' in both conditions.

Histogram bars grow longer with higher accuracy values except for the outlier in the sub-20% range. This shows that the number of low-accuracy results is small and, depending on the definition of 'low-accuracy', may be very small as only 1% of the video sequences were segmented with lower than 60% cross-set intersection accuracy.

Given these results, it is evident that most of the data is segmented with a high accuracy by the Agrast-6 neural network.

4.4.5.4. Qualitative error analysis

Some examples of errors made by Agrast-6 are shown in Figure 4.27. The green color in the images represents the ground truth, the red color is Agrast-6 output, and the yellow color shows their intersection (100% accuracy would be shown as yellow-only shape). The first two images show extreme examples of over-segmentation. The neural network only captured a small part of the human body (16% and 31%, respectively).

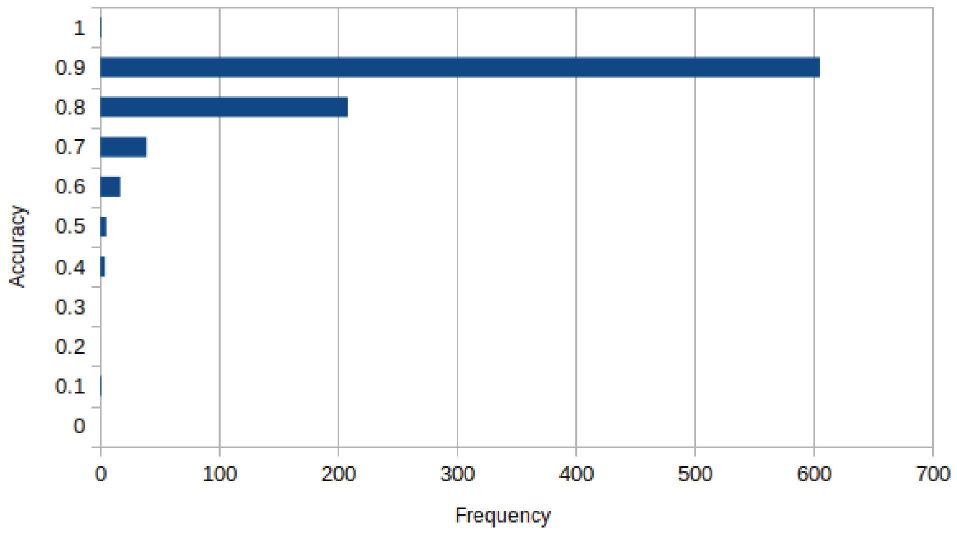


Figure 4.26. Agrast-6 segmentation accuracy histogram

The person in the second image is sitting on a chair that partly occludes the human body. This might have contributed to the low segmentation accuracy. The third image shows an example where the human is wearing black jeans – the depth image is very noisy around this area, and it was difficult for the network to learn about the 'invisible' legs.

The final three images included some shapes that are similar to a human. There was a chair on the left side of the image, and it somewhat resembles the shape of a human. Even though objects like this were present in the training dataset, the network sometimes cannot distinguish them from a real human shape. In the last image, there was another human in the scene, and the network had no means to learn what is the 'main' human in the scene as it captured both humans. These errors could be resolved by a post-processing step which removes the smaller objects from the segmentation output.

It is, however, worth noting that the 5 initial images belong to the bottom-1% accuracy images, and the final image is from the bottom-7%. Therefore, mistakes like these are not very common. The most common accuracies are shown in Figure 4.28. The human shape is segmented correctly as the errors are hardly visible to the naked eye. However, they occur due to some noise in the image and around the boundaries of the silhouette. As stated above, the upsampling layers of the network find it difficult to correctly predict the exact boundaries of the encoded silhouette, therefore, some errors occur when deciding where the boundary is. This is evident for all 6 provided images, and this is the only reason preventing the 100% accuracy. However, the current segmentation accuracy is acceptable for at least some applications.

Figure 4.29 shows the best segmentation of the network. The human is segmented with high precision, however, there is still some fuzziness around the edges of

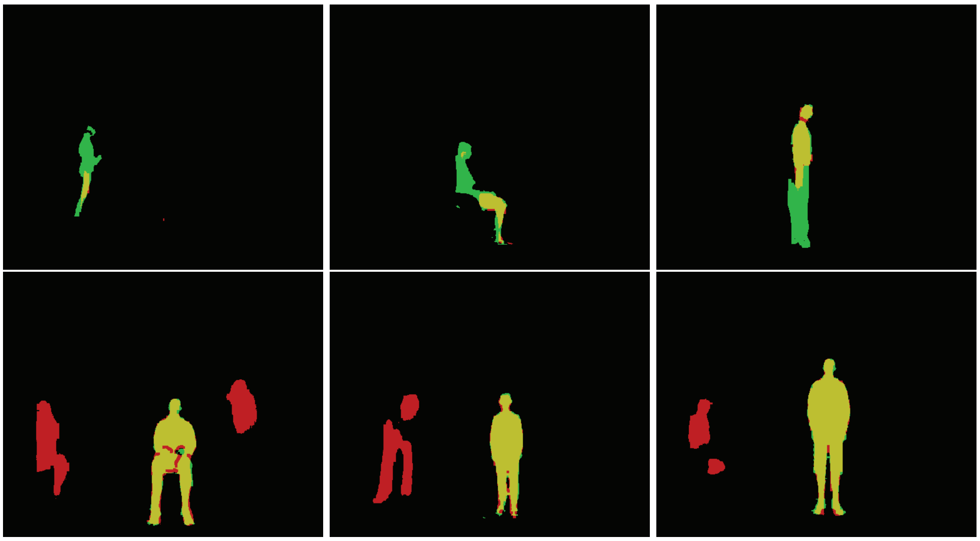


Figure 4.27. Examples of low-accuracy segmentation outputs

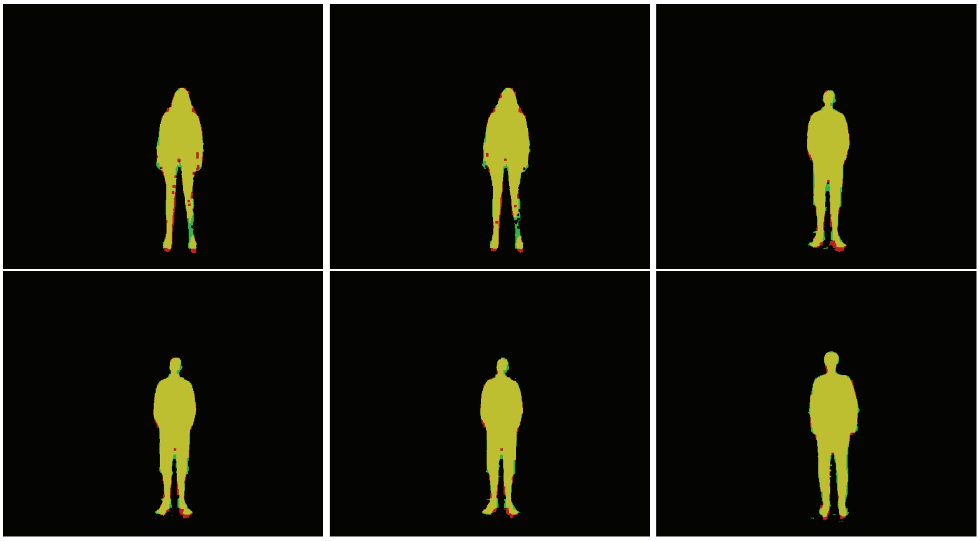


Figure 4.28. Examples of most typical segmentation accuracy images. Accuracies of 86%, 88%, 89%, 90%, 91% and 92%, respectively

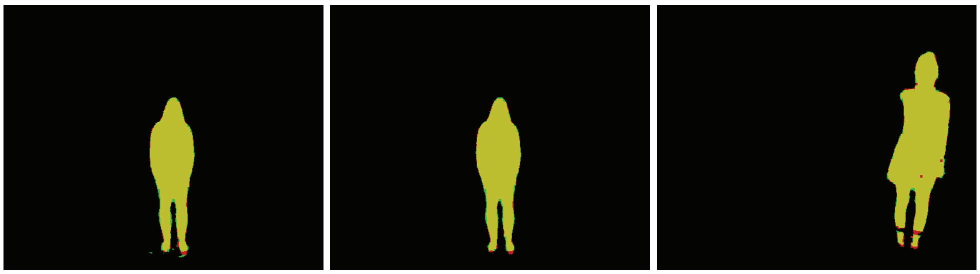


Figure 4.29. Examples of the best segmentation outputs (93% and above)

the silhouette. The Agrast-6 network is usually imprecise by a few pixels. The shape of the human is preserved, and there are only a few artifacts on the output due to noise in the image. The first image even has some invalid ground truth to the left of the human foot, and the neural network correctly did not find it as a part of the human body. These examples all come from the test dataset – they have not been used to train the network. This further confirms that the network has not overfit the data as the output on unseen data is usually correct.

4.4.5.5. Percentage of well-segmented frames

Figure 4.30 shows frame percentages by segmentation accuracy. The green line represents the simple dataset, whereas the blue line shows the complex dataset. Both curves stay very high until about 60% accuracy threshold, which means that there is a very high chance that the segmentation accuracy will be above this mark. There is a steep decline in the frame percentage in the 80-92% range for the simple dataset and the 83-94% range for the complex dataset. Therefore, if the minimum acceptable accuracy is around 80%, Agrast-6 will work in most cases. If the threshold is 85%, the complex dataset will have a much better accuracy than the simple dataset. The 90% threshold is also a viable mark, however, the percentage of frames will be somewhat lower at about 69% for both datasets combined.

The curve has a desirable shape where it stays high as long as possible, and only drops with high accuracy values. Errors at 90% accuracy are already not easily visible to a naked eye, thus the results suggest that Agrast-6 can be applied for binary segmentation as successfully as the other state-of-the-art solutions, but the network itself is much smaller, which confirms the hypothesis that binary segmentation can be successfully solved with a relatively high accuracy and a small architecture.

Table 4.17 shows the pixel-wise confusion matrix for Agrast-6 when applied to the simple dataset. The true positive rate is 90.9% while the true negative rate is 99.8%. These numbers show that the neural network misses some pixels of the human while the background pixels are labeled correctly with better accuracy, however, both rates are above the 90% mark. The pixel-wise accuracy of the model on this dataset is 99.4%, which shows that a vast majority of all points are classified correctly, and it already becomes difficult for the network to increase it any further. The precision and recall values are lower at 92.2% and 90.3% respectively because the amount of the true neg-

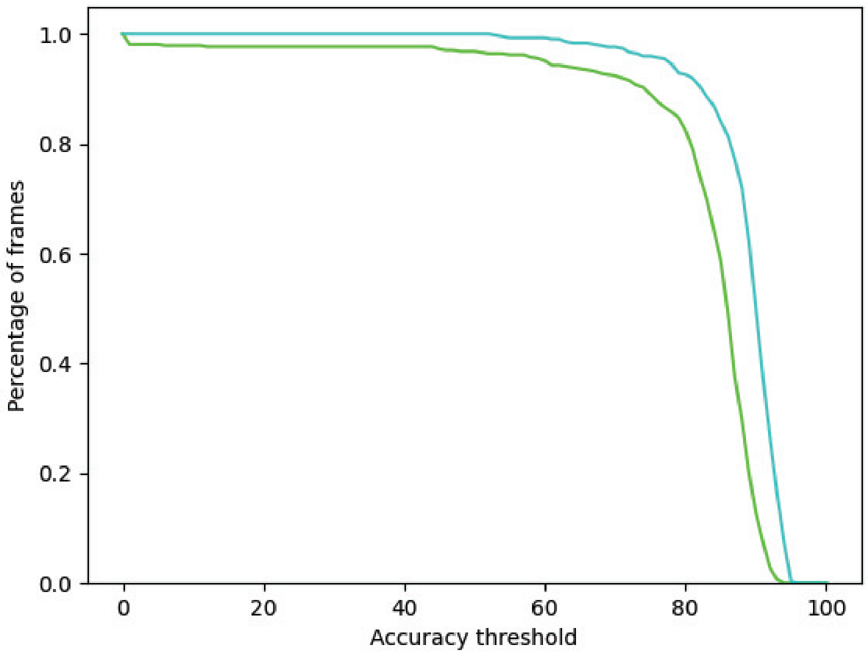


Figure 4.30. Agrast-6 accuracy frame percentages

Table 4.17. Pixel-wise confusion matrix for Agrast-6 segmentation (simple dataset)

		Predicted		
		Background (negative)	Foreground (positive)	Total
Actual	Background (negative)	96.4%	0.3%	96.7%
	Foreground (positive)	0.3%	3.0%	3.3%
Total		96.7%	3.3%	100%

Table 4.18. Pixel-wise confusion matrix for Agrast-6 segmentation (complex dataset)

		Predicted		
		Background (negative)	Foreground (positive)	Total
Actual	Background (negative)	93.2%	0.3%	93.5%
	Foreground (positive)	0.4%	6.0%	6.4%
Total		93.6%	6.3%	100%

atives (as well as the amount of the actual negatives) is dominant in the data. Despite this, all pixel-wise segmentation quality metrics are above 90%, which is on par with the state of the art.

Similarly, Table 4.18 outlines the pixel-wise confusion matrix for the complex dataset. The true positive and the true negative rates are similar to the simple dataset at 93.8% and 99.7%, respectively. The accuracy, however, is slightly lower at 99.2% as there are slightly more false negatives. The precision and recall are both higher at 94.6% and 93.2%, respectively, which shows that the human is segmented better. This may be explained by the size of the human silhouette in this dataset – there are more positive pixels in the data, hence making the same amount of errors results in lower error rates, and, consequently, better precision and recall. It is shown in Section 4.4.3 that the neural network finds it difficult to exactly find the edge pixels. Since the number of such pixels grows slower than the total amount of positive pixels when the human comes closer to the camera, it affects the precision and recall metrics. Therefore, the two confusion matrices confirm that the output of Agrast-6 is generally correct with uncertainty around the object edges.

The accuracy of the Agrast-6 neural network is also visualized by the receiver operator characteristic (ROC) curve in Figure 4.31. There is a very steep increase at the start of the graph which rises to a value of about 0.98 true positives while false positives stay near 0. The curve then smoothly ascends until it reaches the value of 1. The area under curve (AUC) is equal to 0.9903. This shows that the model classifies pixels with a high accuracy – there are very few instances where a positive pixel would be classified

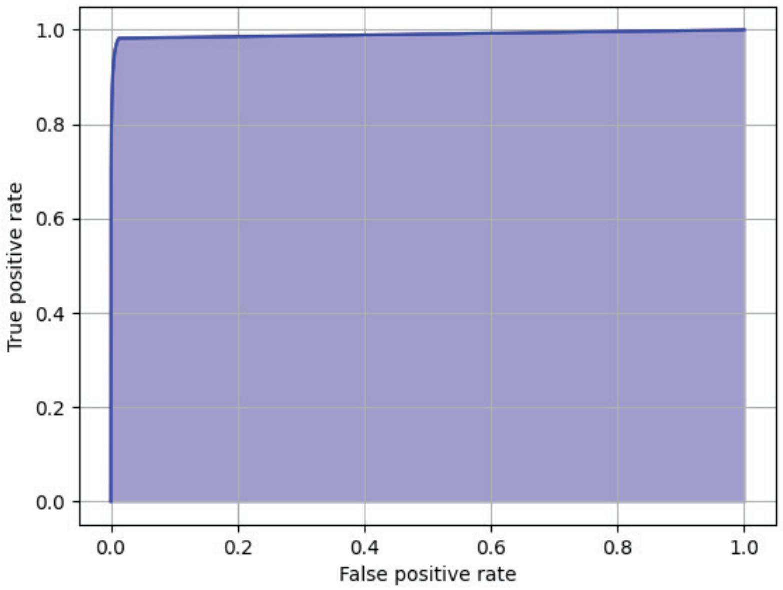


Figure 4.31. ROC curve for Agrast-6 neural network

with a lower score than a negative pixel (this probability is equal to $1 - AUC = 0.0097$). The curve was computed from the classification output of the test dataset as the network has not seen the samples. This is a further confirmation that the network performs well on new data.

4.4.5.6. Accuracy analysis by the sex of the subject

Female segmentation was performed with a better accuracy on average as shown in Table 4.19. This follows the same pattern as seen in the previous chapters as female silhouettes are segmented with a better accuracy than male silhouettes across both datasets. The reasons for this could be the same as women's clothes, such as dresses, are easier to recognize for the neural network as they have a smoother shape compared to the other types of clothes. This has a strong impact since the complex dataset contains more samples with male subjects. The simple dataset, on the other hand, contains an equal number of male and female subject samples, and the difference in accuracy is larger than with an imbalanced dataset.

4.4.5.7. Comparison against *SegNet* neural network

Agrast-6 is a lightweight modification of the *SegNet* neural network [50] with fewer and smaller layers and dropped skip connections. Therefore, these two models are compared in terms of their static and execution parameters. Agrast-6 is built by using TensorFlow, therefore, *SegNet* was also adapted from an already existing TensorFlow implementation [346] of *SegNet* neural network. It was also adapted to the

Table 4.19. Male vs. female detection accuracy

Dataset	Gender	Mean cross-set intersection
Simple	male	82.2%
Complex	male	86.7%
Simple	female	85.3%
Complex	female	87.3%

Table 4.20. Comparison of *SegNet* and the proposed Agrast-6 neural networks

	SegNet	Agrast-6
Model size on disk	117 MB	15.4 MB
Parameter count	32M	1.25M
Inference time	69 ms @ 340x480	34 ms @ 448x512
Average cross-set intersection	89.5%	85.4%

binary data used in this research – the amount of output labels is reduced from 12 to 2. It was then trained by using the same data as Agrast-6 to have an as close comparison as possible. *SegNet* and Agrast-6 both use Adam optimizer, however, the training rates are different (0.001 for *SegNet* and 0.0001 for Agrast-6) – these values have not been modified. When exported to the disk, *SegNet* takes up about 117 MB of the disk space, while Agrast-6 is about 7.5 times smaller at 15.4 MB. The amount of parameters is 27 times smaller for Agrast-6 as well. The inference time was measured by using the TensorFlow 2.8 library on NVidia GTX 1660 SUPER GPU, however, the images were downscaled for *SegNet* to its original 340x480 resolution while Agrast-6 processed unmodified 448x512 resolution images. Binary masks were predicted one at a time (batch size = 1). The inference time was 69 ms per frame for *SegNet* and 34 ms for Agrast-6. The time was measured for the inference only, and any required preprocessing is not taken into account. This shows that Agrast-6 is about 2 times faster with 40% larger images. The speedup is expected since Agrast-6 is a much smaller architecture. However, this comes at a cost of accuracy – *SegNet* processes data consistently more accurately. The difference is similar across all accuracy benchmarks, and the average is the 4.1 percentage points higher cross-set intersection over Agrast-6 when both datasets are weighted equally (89.5% for *SegNet*, 85.4% for Agrast-6). This shows that Agrast-6 carries a lot more accuracy per parameter, and, while sacrificing some overall accuracy, works twice as fast. The summary is outlined in Table 4.20.

4.5. Skeleton transformation and fusion experimental evaluation

4.5.1. Skeleton transformation and fusion accuracy experimental evaluation

A separate dataset of at least 2400 frames from three *Kinect 2* sensors (800 each) has been used for accuracy evaluations. The selected environment was a spacious room with three sensors 4 meters apart monitoring the same central space. Artificial light was used in the room. The captured humans were ordered to stand as still as possible to obtain static skeletons.

Table 4.21. Buffered values fusion method accuracy impact evaluation

	Average	Median
$D(x)$	0.042	0.032
$D(y)$	0.010	0.015
$D(z)$	0.027	0.050
MD	0.026	0.032

Table 4.22. Buffer size accuracy impact evaluation

	1	50	100
$D(x)$	0.013	0.056	0.032
$D(x)$	0.014	0.012	0.009
$D(x)$	0.022	0.024	0.021
$D(x)$	0.016	0.031	0.021

The test system collects skeletal data from each *Kinect* sensor and applies the transformation to a common coordinate space described in Sections (3.28)-(3.34). The coordinates of the skeleton joints would be equal in a perfect world. In real-life scenarios, however, they are different between the different sensors. Two methods of skeleton fusion are evaluated – the average and median values which are described in Section 3.7.2. Accuracy is evaluated by joint fluctuations as this method is considered more accurate if the values over time are less dispersed. Dispersion is used as a quantitative measure. Dispersion is computed for each joint separately, and the average of dispersions is computed.

First, the fused skeleton joint coordinates can be either averaged, or a median can be used. The results by dimension are shown in Table 4.21. The median is more accurate in the x coordinate (left-right direction), while the average gives better accuracy in the y (up-down direction) and z (depth direction) coordinates. The fluctuations are the smallest in the y direction in both cases. The z direction was much more accurately evaluated by using averages than medians. The results suggest that either the average should be used, or a combination of methods should be applied – the median computed for one coordinate, and the average computed for the others. When using the average, the standard deviation is 0.16, which is ~2%.

Next, the buffer size impact was evaluated. The buffer sizes of 1, 50, and 100 were selected for evaluation. The results have shown that buffering does not improve the accuracy as expected. This means that buffering values are not a viable solution since it reduces both accuracy and responsiveness. The detailed results are outlined in Table 4.22. These results may have been obtained as such because the person does not stand in a perfectly static position and moves a little. These small movements may impact the accuracy negatively.

The final hypothesis to check is that rejecting points reported with low confidence may improve the accuracy in combination with other techniques. However, this was

Table 4.23. Low confidence point accuracy impact evaluation

	With low-confidence	Without low-confidence
$D(x)$	0.056	0.412
$D(y)$	0.012	0.051
$D(z)$	0.028	0.588
MD	0.031	0.350

also shown not to be the case. The fluctuations with low-confidence points excluded were much higher, as shown in Table 4.23. It was observed that some joints 'jump' in the *Kinect* skeleton output between the frames even though the person is not moving. These joints may be misclassified by *Kinect*, however, these fluctuations are averaged-out when the data from all sensors are included.

It was shown in related research that fluctuations in the standing poses when using a single *Kinect* device reach about 10% [37]. The results shown in this section suggest that the use of a triple *Kinect* setup reduces this number to ~2%, which confirms that the transformation and fusion algorithms improve the accuracy of skeleton tracking.

4.5.2. Skeleton transformation and fusion performance experimental evaluation

The transformation and fusion algorithms proposed in Section 3.7.2 do not depend on the data in terms of performance. Therefore, any skeletal data can be used to evaluate the performance of skeleton transformation.

The benchmark was performed by using frame sequences of about 300 frames from three cameras and by using the first frame for the detection of transformation parameters, and then, by using those parameters, to transform the skeleton in the other frames. The implementation uses the C# programming language, .NET Framework v4.5.2. The benchmark was performed on AMD Ryzen 9 3900X CPU. The observed average processing duration was **243 µs** per frame-triplet. This result shows that the processing time of the proposed solution is so small that it may as well not be considered when implementing multi-sensor systems using *Kinect*. *Kinect 2* devices have to be connected to different computers due to their technological limitations. This means that the data must be transferred over the network, which typically includes data serialization, send/receive, and deserialization. This extra work typically takes over 1 ms, which is a much longer duration than the transformation/fusion itself.

4.6. Summary

Two datasets have been collected for segmentation research purposes. The 'simple' poses dataset consists of 266 short depth video sequences, whereas the'complex' dataset contains 674 depth video sequences. They were used to evaluate the segmentation algorithms proposed in this dissertation.

First, the noise reduction algorithm experimental analysis was conducted. It revealed that the bilateral filter has a better effect on the segmentation accuracy than the median, the Gaussian filter, or no filter at all, however, it is the slowest filter.

The experimental analysis of the proposed improvements for semi-automatic segmentation algorithms showed that all proposals improve segmentation performance by large margins. The original solution of the PCL library, Euclidean clustering, and its re-implementation in the Java language were used as baselines. The Java variant was about 2 times slower than the original C++ variant (2.3 s vs. 1.2 s). All other times were measured for the Java implementations of algorithm variations. First, Euclidean clustering can run about 4 times faster when the branch cutting improvement is implemented, about 38 times faster with the node removal, and about 44 times faster with both improvements. Switching from the Euclidean clustering approach to the bounding-box-based segmentation shows a speedup of 63%. The bounding-box-based segmentation with the node removal, branch cutting, and auto-expanding bounding box improvements was about 225 times faster than the bounding-box-based approach with no improvements, 195 times faster than C++ Euclidean clustering implementation, and 367 times faster than Euclidean clustering Java implementation.

The Euclidean-clustering-based and bounding-box-based segmentation algorithms yield similar accuracy for the simple dataset (79.8% and 82.1%, respectively), however, the Euclidean-clustering-based segmentation is more accurate for the complex dataset segmentation (48.1% vs. 33.4%). Both algorithms tend to make under-segmentation errors in complex scenes, however, the bounding-box-based algorithm is more unstable in this sense. The Euclidean-clustering-based algorithm yielded accuracies between 30% and 80% much more often than the bounding-box-based approach. Despite their imperfections, both algorithms can reduce the total time required to prepare datasets for segmentation. It was estimated that a fully-manual workflow would take 1455 hours of work, while the software solution, utilizing the bounding-box-based segmentation algorithm, reduced this time to 25 hours. Euclidean-clustering-based segmentation was estimated to require 28 hours due to the longer processing times, despite the improved accuracy.

The recursive 2-Means split algorithm improved the accuracy compared to the original bounding-box-based segmentation algorithm. The accuracy went up from 82.1% to 82.5% for the simple dataset and from 33.4% to 55.3% for the complex dataset. The latter is a big improvement as it even beats the Euclidean-clustering-based segmentation accuracy. This comes at a performance cost, and processing times go from 6.3 ms to 46.9 ms per frame. It was estimated that it would have reduced the total dataset preparation time from the current 25 hours to 18.5 hours due to reductions in manual work.

The *Agrast-6* model was trained using the annotated datasets. It showed progress via the binary accuracy, precision, recall, and AUC metrics, as well as the loss function for both training and validation datasets, with all values improving. This shows that the network does not overfit the data and can learn from the provided data. It was noticed that the most difficult parts for the network to learn are the human head and feet, while the body is segmented with good accuracy. The simple dataset segmentation accuracy was 82.1%, whereas the complex dataset scored 88.6%. Error analysis showed that there are virtually no very low accuracy (below 40%) cases, and AUC of 0.9903 was achieved. The model is lighter than the baseline *SegNet* (117 MB on the disk vs. 15.4 MB) and processes images faster (69 ms vs. 34 ms, however, the *Agrast-6* image size was larger). On the other hand, *SegNet* was more accurate (89.5% vs. 85.4%).

Rigid body part coordinate fluctuations in the skeletal data were reduced from 10% when using a single *Kinect* device to 2% when using three devices and the proposed skeleton fusion algorithm. It was concluded that the best option for skeleton fusion is the use of the mean coordinate, and it is not worth buffering and averaging skeletons as this does not improve the accuracy.

5. SUMMARY AND CONCLUSIONS

The overview of the state of the art has revealed that the binary depth segmentation problem could be solved by using neural networks for RGB image segmentation. However, these networks require large datasets for training, and obtaining them involves a lot of manual annotations. Semi-automatic segmentation algorithms were reviewed, and novel algorithms were proposed in order to speed up the annotation process. It was shown that the Euclidean-clustering-based approach provides finer control over the segmentation of cluttered scenes, but the bounding box approach is up to 367 times faster. The machine-learning-based segmentation convolutional neural network architectures are usually large, while related research also exposes their low information density. This suggests that smaller architectures should be possible, especially for depth segmentation, since the data is more simple. It was proven correct by the *Agrast-6* neural network proposal, which achieved a comparable accuracy as the state-of-the-art models with a smaller architecture. The skeletal data provided by the *Kinect* sensor is noisy and fluctuating, and this problem is often solved by utilizing multiple sensors. Therefore, it was decided that a novel algorithm for skeleton fusion should be proposed. It successfully reduced the fluctuations in the skeletal data when using 3 sensors compared to using a single sensor.

Conclusions of the dissertation:

1. Literature review has revealed that both geometrical and machine learning methods can be applied to binary segmentation problems, however, the machine-learning-based methods usually offer a higher accuracy at a cost of a large amount of data required for training, which leaves dataset preparation as one the biggest bottlenecks in training segmenting neural networks.
2. Both the Euclidean-clustering-based and the proposed bounding-box-based segmentation algorithms fit the problem of the semi-automatic binary depth segmentation. The Euclidean-clustering-based algorithm offers a better accuracy, especially in cluttered scenes (48.1% vs. 33.4%), but the bounding-box-based segmentation performs up to 367 times faster (2.33 s vs. 6.33 ms). The bounding-box-based solution reduced the amount of the required human time from estimated 1455 hours of full manual work to 25 hours.
3. Recursive 2-Means split algorithm with a random forest classifier for split acceptance improves the segmentation accuracy where it was originally low (33.4% vs. 55.3%) at a cost of lower performance (6.33 ms vs. 40.6 ms). This confirms that over-segmentation is reduced by using the proposal. However, the accuracy could be further improved if a better point cloud split algorithm could be found instead of the proposed recursive 2-Means split.
4. The proposed semi-automatic segmentation algorithms greatly reduce the time required to label binary masks for depth video sequences. They allowed preparing ground-truth masks for 220k images in 25 human hours by using the proposed software solution, whereas state-of-the-art datasets for similar purposes

often require entire teams to prepare.

5. The proposed *Agrast-6* architecture shows that smaller architectures may yield a similar accuracy for binary depth segmentation as the large architectures meant for RGB segmentation. *Agrast-6* has 27 times fewer parameters to train compared to the *SegNet* neural network, but still reaches 86% binary segmentation accuracy. Despite that, the analysis has shown that the encoder of the proposed architecture has few activations, and, therefore, it should be possible to reach a similar accuracy by shrinking the encoder.
6. The proposed skeleton fusion algorithm reduced the fluctuations in the skeletal data from 10% originally, when using a single *Kinect* camera, to 2% when using three cameras and the algorithm. It was successfully used in further research to evaluate the load of human joints during physical activity.

6. SANTRAUKA

6.1. Įžanga

6.1.1. Darbo aktualumas

Taškų debesų segmentavimas yra pastaruoju metu aktyviai tiriama sritis. Nors įprastos vaizdo kameros yra plačiai prieinamos, gylio jutikliai turi savų privalumų. Vis dažniau kylant privatumo problemoms, RGB kameros fiksuoja daug duomenų ir dėl to kelia saugumo grėsmes [1]. Gylio duomenys turi mažiau jautrios informacijos, ypač kai kalbama apie žmogaus stebėseną. Gylio duomenų segmentavimas gali būti taikomas daugelyje sričių. Jis dažniausiai yra vienas iš komponentų taikant praktikoje. Tai yra viliojanti tyrimų sritis, nes pigios gylio kameros [2] bei lazeriniai lokatoriai [3] yra plačiai prieinami. Didelė tyrimų sritis yra erdvinių duomenų, pateikiamų gylio jutiklių, segmentavimas [4]. Jie naudojami daugelyje tyrimų sričių – trimačiam veido atpažinimui [5], kritimui aptikti [6], viršutinių galūnių charakteristikoms įvertinti [7], taikant fitnesą [8, 9], pratimams instruktuoti [10], pramonės darbuotojų aktyvumui stebėti [11], robotikoje [12], kliūtims aptikti neregiams [13], antropometriniams matavimams [14], laikysenai atpažinti [15] arba bendrajam žmogaus kūno sekimui [16] ar net paveikslėliams šifruoti [17]. Tačiau išlieka ir daug problemų apdorojant gylio duomenis, priklausomai nuo taikymų srities.

Objektų segmentavimo uždavinys patraukė daugelio tyrėjų dėmesį, ypač atsiradus mašininio mokymo sprendimams [18]. Vienas iš kertinių šio proceso komponentų yra etikečių priskyrimas taškeliams. Etiketės dažniausiai reiškia objekto, kuriam priklauso taškelis, tipą [19]. Giliojo mokymosi metodai, ypač sąsūkos neuroniniai tinklai, plačiai naudojami objektų segmentavime. Jie daug prisidėjo prie šios srities tyrimų skaičiaus. Vienas iš iššūkių yra suprasti paveikslėlius semantiniu lygmeniu, tačiau tik naujausi sprendimai yra tinkami šiai problemai spręsti [20]. Vienos klasės (dvejetainis) segmentavimas taip pat kartais yra naudingas. Jis taikomas tokiose srityse, kaip debesų segmentavimas [21], medicininiai paveikslėliai [22] ar žmogaus kūno segmentavimas [23].

Jeigu yra žinoma, kad tam tikras objektas jau yra scenoje, vienintelė neišspręsta problema yra šio objekto išskyrimas iš fono. Tai galima atlikti naudojant esamas semantinio segmentavimo sistemas, tačiau jos dažnai yra sudėtingesnės, nei iš tiesų reikia. Todėl jas yra sunkiau išmokyti. Šias problemas pripažįsta Šazeras ir kt. [24], kurie siūlo sprendimą dalį tinklo išjungti, arba Huangas ir kt. [25], kurie siūlo sprendimą efektyviau išmokyti labai didelius tinklus. Taikymų sritys taip pat yra labai įvairios [26]. Kita vertus, moderniausi tinklai yra labai sudėtingi. „VGG-16" yra tinklas, skirtas paveikslėliams klasifikuoti. Jis yra daugelio modernių paveikslėlius segmentuojančių tinklų pagrindas. Šis tinklas turi 133 mln. mokomų svorių [27]. Tai reiškia, kad tokiam tinklui išmokyti reikia labai daug paveikslėlių. Originalus tyrimas naudoja 1,45 mln. paveikslėlių išmokyti, testuoti ir validuoti. Kadangi tai prižiūrimo mokymosi metodas, visi paveikslėliai turi būti sužymėti bent dalinai žmogaus. Segmentavimo uždaviniams reikalingas daug didesnis žmogaus darbo indėlis nei tik priskiriant klases paveikslėliams, nes reikia pažmyėti dalį paveikslėlio, kur yra norimas objektas. Dideli neuroniai tinklai taip pat dažnai pasižymi didele apdorojimo trukme. Naujausi tyrimai rodo, kad šiuolaikiniai grafiniai procesoriai gali apdoroti tik nedidelius (iki 256×128 raiškos) paveikslėlius realiu laiku, o klasikiniai neuroniniai tinklai užtrunka iki 180 ms vienam paveikslėliui [28]. Didesniems paveikslėliams reikia daugiau išteklių, pvz., „SegNet“ tinklas 1920×1080 raiškos paveikslėlį apdoroja per 637 ms naudojant „NVIDIA Titan X“ grafinį procesorių. Tačiau gali būti tinkamos ir mažesnės architektūros [29]. Jos gali greičiau apdoroti didelius paveikslėlius. Tai leidžia tokias architektūras taikyti naujose srityse – jos gali apdoroti duomenis realiu laiku, sumažinti aparatūrinius reikalavimus, pagerinti energijos efektyvumą, joms reikia mažiau mokymosi duomenų.

Duomenų rinkinius, tinkamus paveikslėliams segmentuoti, dar sunkiau paruošti. Nohas ir kt. naudojo 12,3 tūkst. paveikslėlių rinkinį [30] savo „Deconvnet“ architektūrai išmokyti. Duomenų rinkinio pavadinimas yra „PASCAL“, jis tyrimų metu jau buvo 5 metų senumo [31]. Jis buvo surinktas pasirinkus tam tikrus internete pasiekiamus paveikslėlius ir tada juos rankiniu būdu sužymint pasitelkus žymėtojų komandą. Kadangi duomenų rinkinys nėra didelis, „Deconvnet“ mokymas tapo sudėtinga procedūra. Buvo naudojamas partijos normalizavimas, kad tinklas pabėgtų iš lokalių minimumų, ir kuriamas papildomas „PASCAL“ duomenų rinkinio poaibis mažinant erdvinius svyravimus, kad tinklas lengviau pagautų reikiamas objektų savybes ir tik tada mokytųsi erdvines transformacijas. Jeigu duomenų rinkinius būtų lengviau paruošti segmentuoti, „PASCAL“ komanda būtų galėjusi sukurti didesnį duomenų rinkinį, ir dėl to būtų buvę lengviau išmokyti „Deconvnet“ arba taikyti metodai būtų buvę efektyvesni dėl didesnio išmokyti skirtų paveikslėlių rinkinio. Kitas pavyzdys – Xu ir kt. naudojo 43 tūkst. anotuotų nuskaitymų savo tyrimams [32]. „ShapeNet“ duomenų rinkinį sudaro 3 mln. modelių, tačiau išleidimo metu tik 220 tūkst. buvo sužymėti [33].

Duomenų rinkiniai, skirti segmentuoti iš vaizdo įrašų, yra dar imlesni darbui. Pavyzdžiui, šiuolaikinių kamerų kadrų dažnis yra 30 kadrų per sekundę. Vienos minutės ilgio įrašą sudaro 1800 kadrų. Jei tokius duomenis reikia paruošti neuroniniam tinklui mokyti, reikia praktiškai neįmanomo kiekio rankinio darbo objektų kaukėms pažymėti. Tačiau tokie duomenų rinkiniai reikalingi tam tikriems taikymams, pvz., žmogui segmentuoti realiu laiku naršyklėje [34].

Išsegmentuotas žmogaus kūnas gali būti naudojamas kaip komponentas didesniame duomenų apdorojimo procese. Segmentuoti taškų debesys naudojami aproksimuoti žmogaus pozų tikimybėms [35]. Gylio paveikslėliai buvo sujungiami į trimačius žmogaus kūno modelius, o vienas iš žingsnių buvo surasti žmogaus kūną [36].

„Kinect“ jutiklis taip pat pateikia savo skeleto informacijos išvestis su 25 sąnariais, tačiau jos daugeliu atvejų nėra tikslios [37]. Toks skeletų srautas tinkamas daugeliu atvejų, tokių kaip žmogaus sekimo sprendimuose veiklai atpažinti [38, 39], medicinoje, pvz., pusiausvyros normalizavimo pagalbininkuose [40, 41], stovėsenai kontroliuoti [42, 43], Parkinsono ligos asistentams [44]. Taip pat taikoma ir gestams atpažinti [45]. „Kinect“ taip pat naudojamas daugiajutiklėse sekimo sistemose su dėvimais jutikliais, kad padidintų sekimo tikslumą [46]. Visos minėtos taikymų sritys remiasi aukštos kokybės skeleto sekimu. Šią problemą galima spręsti panaudojant keletą jutiklių, tačiau iškyla dvi papildomos problemos. Pirma, kiekvienas jutiklis turi savo koordinačių sistemą, todėl jutiklius reikia kalibruoti [47]. Antra, skeletus reikia sujungti į vieną [48]. Deja, nė viena iš šių problemų neturi vieno geriausio sprendimo, šiuolaikiniai tyrimai siūlo labai skirtingus sprendimo būdus.

6.1.2. Problemos formuluotė

Dvejetainių gylio paveikslėlių segmentavimas yra problema, kai reikia išgauti kaukę gylio paveikslėliui, atitinkančią „priekinį planą“ ir „foną“. Pritaikius žmogaus kūnui sukonstruojama dvejetainė kaukė, nurodanti taškus, priklausančius žmogaus kūnui. Tai svarbu tais atvejais, kai reikia išgauti žmogaus kūną iš paveikslėlio – tada jis gali būti naudojamas žaliesiems ekranams, žmogaus kūnui sekti žaidimams, medicinoje, pvz., reabilitacijai, ar sportui, pvz., jogai. Taikymai nėra šio darbo dalis.

Dvejetainio gylio paveikslėlio segmentavimas gali būti formuluojamas kaip problema, kur reikia gauti dvejetainę kaukę B iš gylio paveikslėlio D:

$$D = S(B), \tag{6.1}$$

čia S yra segmentavimo funkcija ar algoritmas, D – gylio paveikslėlis, B – dvejetainis paveikslėlis su dviem reikšmėmis – objekto taškas arba fono taškas. Funkcija gali būti apibrėžiama skirtingai ir turėti daugiau parametrų priklausomai nuo taikymo srities. Šis tyrimas koncentruojasi į šias problemas bei pateikia karkasą joms spręsti:

- Pusiau automatinis gylio vaizdų segmentavimas.
- Automatinis gylio vaizdų segmentavimas.

Abu segmentavimo tipai gali būti taikomi bendriniam segmentavimui arba konkretaus tipo objektams. Bendrinis segmentavimas neturi informacijos apie tai, koks objektas bus segmentuojamas, dėl to jam reikia papildomų parametrų, suteikiančių informaciją, kokio objekto ieškoti. Automatinis segmentavimas, pagal apibrėžimą, negali turėti tokių parametrų, jis turi surasti objektą tik iš duotų duomenų. Abiem atvejais šis tyrimas koncentruojasi į dvejetainį segmentavimą, kur objektas yra žmogaus kūnas. Darbo akcentas yra pusiau automatinio segmentavimo taikymas duomenų rinkiniams, skirtiems prižiūrimo mokymosi segmentavimo neuroniniams tinklams, paruošti, o automatinis segmentavimas turėtų būti taikomas kaip vienas iš komponentų didesnėje gylio duomenų apdorojimo sistemoje. Kompiuterizuoto pusiau automatinio segmentavimo tyrimas akcentuoja algoritmų greitaveiką, kad prisidėtų prie galutinio tikslo – sumažinti bendrą laiką, reikalingą dvejetainio segmentavimo duomenų rinkiniams paruošti. Automatinis segmentavimas koncentruojasi į mažesnį architektūros dydį, kas leidžia greičiau ją išmokyti bei galimai sumažina duomenų apdorojimo laiką su didesniais paveikslėliais.

Formaliau, kompiuterizuotas pusiau automatinis segmentavimas gali būti apibrėžiamas kaip funkcija su papildomos informacijos parametrais ir algoritmui reikalingais hiperparametrais, palyginti su funkcija S iš formulės (6.1):

$$D = S_c(B, m, h). \tag{6.2}$$

Parametras m yra pateikiamas iš išorės, pavyzdžiui, žmogaus rankiniu būdu, o h yra algoritmo hiperparametrų rinkinys. Automatinis segmentavimas, pagal apibrėžimą, neturi parametro m:

$$D = S_a(B, h). \tag{6.3}$$

Skeletų sąlaja yra kita problema, susijusi su žmogaus kūno sekimu. Ji naudojama panašiuose taikymuose, tačiau skiriasi sekimo principas – vietoj priekinio plano-fono kaukių naudojami iš anksto gauti skeleto duomenys iš žmogaus sekimo įrenginių ir bando pagerinti tikslumą, palyginti su sekimu naudojant vieną jutiklį.

Formaliau, skeletų sąlaja yra problema, kur, turint žmogaus kūną trimatėje erdvėje, stebimą keleto jutiklių iš skirtingų kampų, konstruojamas vienas superskeletas, kuris yra tikslesnis nei naudojant vieną jutiklį. Skeletų sąlaja susideda iš dviejų mažesnių problemų: skeletų transformacijos į bendrą koordinačių sistermą bei skeletų sujungimo, po ko kurio gaunamas vienas skeletas. Tarkime, kad turime skeletų rinkinį iš skirtingų kampų $K = \{K_1, K_2, \ldots, K_n\}$. Tuomet turėtų būti pasiūlytos transformacijos funkcija B ir sujungimo funkcija M, kad būtų gaunamas tikslesnis superskeletas K_s:

$$K_s = M(T(K)). \tag{6.4}$$

Šio darbo tikslas yra:

- pateikti algoritmą S_c, kuris leistų sumažinti bendrą reikalingą laiką dvejetainio segmentavimo duomenų rinkiniams paruošti, palyginti su automatiniu segmentavimu;
- pateikti lengvasvorę neuroninio tinklo architektūrą kaip funkciją S_a su mažesniu parametrų skaičiumi ir trumpesniu duomenų apdorojimo laiku nei neuroniniai tinklai keleto klasių RGB segmentavimui;
- pateikti funkcijas T ir M, kurios sumažintų svyravimus su trijų „Kinect“ įrenginių sąranka, palyginti su vienu „Kinect“ įrenginiu.

6.1.3. Uždaviniai

Darbo tikslui pasiekti keliami tokie uždaviniai:

1. Išanalizuoti esamus sprendimus geometriniam paveikslėlių segmentavimui, mašininiu mokymu grįstam segmentavimui bei skeletos sąlajos algoritmus;
2. Pasiūlyti greitą geometrinio paveikslėlių segmentavimo algoritmą;
3. Pasiūlyti automatinio paveikslėlių segmentavimo mašininio mokymosi modelį;
4. Pasiūlyti naują skeletų sąlajos algoritmą;
5. Įvertinti visų sukurtų algoritmų greitaveiką ir tikslumą.

6.1.4. Mokslinis naujumas

Šiame darbe pristatomas toks mokslinis naujumas:

1. Aprėpties dėžutėmis grįstas segmentavimo algoritmas, 367 kartus greitesnis nei euklidiniu klasterizavimu grįstas su pasirinktu duomenų rinkiniu.
2. Rekursinis 2-vidurkių padalijimo algoritmas su atsitiktinio miško klasifikatoriumi padalijimui priimti, kuris sumažina per mažo segmentavimo paklaidas vidutiniškai 2,5 karto su pasirinktu duomenų rinkiniu.

3. Greitas keleto „Kinect“ įrenginių skeletų kalibravimo ir sąlajos algoritmas, sumažinantis skeletų svyravimus nuo 10% iki 2%.
4. Nauja sutrumpinta sąsūkos neuroninio tinklo architektūra žmogaus kūnui segmentuoti, kuriai reikia 27 kartus mažiau parametrų ir kuri apdoroja 40% didesnį paveikslėlį 2 kartus greičiau nei „SegNet“ neuroninis tinklas [50].

6.1.5. Ginamieji teiginiai

- Bendras žmogaus prižiūrimo taškų debesies segmentavimo laikas gali būti sumažintas 66 kartus pritaikius optimizuotą aprėpties dėžutėmis grįstą algoritmą, palyginti su visiškai rankiniu segmentavimu ir 12%, palyginti su euklidine paieška [49].
- Sutrumpintas dvejetainio gylio segmentavimo neuroninis tinklas gali pasiekti panašų tikslumą, kaip ir dideli šiuolaikiniai neuroniniai tinklai, skirti RGB duomenims segmentuoti.
- Aprėpties dėžutėmis grįsto per mažo segmentavimo rezultato plotas gali būti sumažintas 2,5 karto atkertant dalį taškų debesies, naudojant pasiūlytą rekursinį dviejų vidurkių padalijimą ir pasiūlytas padalijimo metrikas.

6.1.6. Praktinė vertė

Pasiūlyti pusiau automatinio segmentavimo algoritmai su jų optimizacijomis yra realizuoti programinės įrangos įrankyje dvejetainėms kaukėms „Kinect“ gylio paveikslėliams žymėti. Jie padėjo sumažinti bendrą kompiuterizuoto segmentavimo laiką ir tapo įmanoma dviem žmonėms sužymėti duomenų rinkinį, kurio dydis 220 tūkst. gylio paveikslėlių. Dėl to pagrindinis algoritmų taikymas yra duomenų žymėjimas prižiūrimo mašininio mokymosi architektūroms. Šie algoritmai potencialiai galėtų būti taikomi ir neprižiūrimo klasterizavimo uždaviniams. Šis įrankis turi daugiausia mokslinio naujumo, nes jame naudojamos naujos algoritmų variacijos.

Pasiūlyta „Agrast-6“ architektūra gali būti taikoma didesnėje gylio duomenų apdorojimo sistemoje, kur pats tinklas būtų vienas iš komponentų. Vienas iš tokių potencialių taikymų yra žmogaus siluetų palyginimas naudojant Hausdorfo atstumo metriką. Tai galėtų būti taikoma jogos treniruotėse, reabilitacijos veiklose ar kitokiam žmogaus sekimui. „Agrast-6“ modelis yra galutinis šios disertacijos rezultatas, nes visi segmentavimo tyrimai veda prie šio modelio ir jo išmokymo.

Pasiūlytas skeletų sąlajos algoritmas buvo pritaikytas žmogaus judesių analizei [9]. Atlikti tikslesni matavimai leido nustatyti žmogaus sąnarių apkrovas atliekant skirtingas veiklas. Algoritmas taip pat galėtų būti taikomas kaip būdas užfiksuoti patikslintą žmogaus skeleto reprezentaciją nauojant kelis „Kinect“ įrenginius.

6.2. Literatūros apžvalga

6.2.1. Gylio duomenų surinkimas ir apdorojimas

6.2.1.1. Gylio jutikliai

„Microsoft Kinect“ įrenginiai teikia RGB, gylio ir IR vaizdo srautus, taip pat žmogaus kūno indekso srautą bei skeleto sąnarių padėtis. Yra trys įrenginio versijos

– „Kinect“, „Kinect 2“ ir „Azure Kinect“. Pastarasis tiksliausias [61, 65], bet nestačiakampis matymo laukas sunkiai panaudojamas [55] Ankstesniųjų vaizdai triukšmingesni [52, 53, 54]. „Kinect“ programinė įranga taip pat nėra tiksli [70, 37, 73], tačiau tai galima dalinai išspręsti [70, 72]. „Kinect“ fiksuoja gylio duomenis apšviesdamas sceną infraraudonaisias spinduliais [76] ir naudodamas skrydžio laiko metodą [77].

6.2.1.2. Skeleto duomenų iš keleto jutiklių sąlaja

Dažniausiai naudojami algoritmai kelių jutiklių duomenims transformuoti į bendrą koordinačių sistemą:

- Iteracinis artimiausio taško algoritmas [98];
- Transformacijos matricomis [102];
- Žymekliais objektais paremtos transformacijos [104];
- Posūkis matricomis [105];
- Metodų derinys [107].

Žymekliais paremtas metodas yra greitesnis, nes nereikia nei apskaičiuoti tinkamų matricų, nei vykdyti keleto iteracijų skaičiavimų, tačiau reikalingas visiems jutikliams matomas specialus objektas. Transformuoti skeletai suliejami į vieną naudojant tiek paprastą vidurkį [105], tiek vertinant patikimumo lygius [99, 102] ar panaudojant ir papildomą gylio informaciją [109].

6.2.1.3. Gylio žemėlapiai

Gylio žemėlapis yra į paveikslėlį panašus gylio vaizdas, plačiai naudojamas su sąsūkos neuroniniais tinklais semantiniam gylio vaizdų segmentavimui [112, 32, 113], tačiau jiems taikomi ir klasikiniai vaizdų aprodorijo algoritmai [116, 117]. Gylio žemėlapiai dažnai naudojami kartu su kitų tipų duomenimis, pvz., RGB ir terminiais vaizdais, kurie gali padidinti sprendimų tikslumą ir kartu sudėtingumą [124, 120, 132]. Naudojant tik gylio duomenis rezultatai mažiau priklauso nuo išorinių faktorių [137, 138].

6.2.1.4. Trimačiai dvejetainiai paieškos medžiai

Dvejetainiai paieškos medžiai yra naudingi taškų debesų analizei, nes paieškos algoritminis sudėtingumas yra $O(\log n)$. Šie medžiai dažnai naudojami segmentavimui [140, 113, 141], klasterizavimui [142, 143, 144]. Subalansuoto daugiamačio medžio kūrimas yra algoritmiškai sudėtingas, nes reikia daug kartų rasti medianas. Tai galima spręsti taikaint trijų medianos algoritmą [149], išankstinį duomenų surikiavimą [150] ar taikyti grafinius procesorius [151].

6.2.1.5. Triukšmo mažinimas

Gylio kadrams tinka tie patys triukšmo mažinimo algoritmai, kaip ir RGB vaizdams. Vidurkio ir medianos filtrai yra paprasti metodai, tačiau gali sulieti kraštus [165, 166]. Gauso filtras efektyvesnis, tačiau sudėtingesnis skaičiavimo požiūriu. Dvišaliai filtrai prideda Gauso filtrų diapazono svorį, geriau išsaugo kraštus [179] ir efektyviai sumažina aukšto dažnio triukšmą [182], bet apskaičiuojami dar lėčiau [187]. Mašininiu mokymusi pagrįsti sprendimai taip pat mažina tam tikrų tipų triukšmą [188], tačiau jiems reikalingas sudėtingas mokymosi procesas ir jie ne visada tinkami.

6.2.2. Taškų debesų panašumo metrikos

Daiso koeficientas (DSC) [189] ir Žakardo indeksas [190] yra dažnai naudojamos metrikos aibių panašumui įvertinti vaizdų segmentavime. Pirmasis išreiškiamas formule:

$$DSC = \frac{2|A \cap B|}{|A| + |B|}, \tag{6.5}$$

antrasis:

$$J = \frac{|A \cap B|}{|A \cup B|}. \tag{6.6}$$

Abi metrikos yra jautresnės per mažam nei per dideliam segmentavimui [192].

6.2.3. Segmentavimo ir klasterizavimo metodai

6.2.3.1. Geometriniai segmentavimo ir klasterizavimo metodai

Euklidinio klasterizavimo [49] veikimo principas parodytas 6.2.1 pav. Jis naudoja spindulio paiešką taškams tam tikru atstumu nuo pradinio taško rasti ir priskiria juos grupei. Algoritmo sudėtingumas yra $O(n \log n)$. Aprėpties dėžutės taip pat taikomos segmentavimo užduotims kaip neuroninių tinklų išvestis [200, 201], tačiau šiems sprendimams išmokyti reikia duomenų rinkinių. Tinkami rinkiniai yra KITTI [207] ir iSAID [208], tačiau jų kūrimas užtrunka dėl didelio kiekio rankinio darbo.

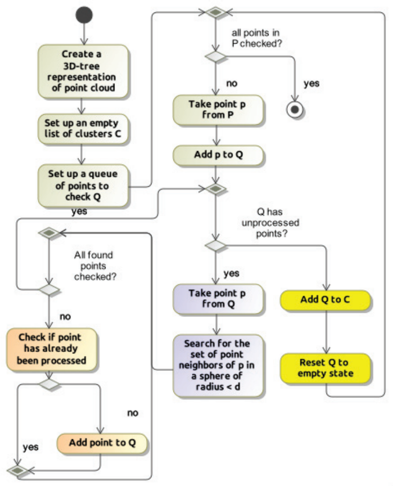
Aprėpties dėžutės gali būti naudojamos klasterizavimui kaip jau esamo klasterio savybė. Jos yra paprastos [212] ir greitai apskaičiuojamos [213, 214], tačiau gali būti netikslios [217] ir ribotos [218], jei reikia segmentuoti sudėtingų formų arba arti esančius objektus. Klasterizavimo metodai gali būti pritaikyti ir segmentavimui [220, 221, 222]. Euklidinis klasterizavimas naudojamas taškams sujungti į segmentus [223] arba padalinti klasterį į segmentus [224].

6.2.3.2. Neprižiūrimas klasterizavimas

K-vidurkių algoritmas [236] yra neprižiūrimo mašininio mokymosi metodas, tinkamas taškų debesims klasterizuoti. Jis turi vieną hiperparametrą k, kuris apibrėžia taškų debesyje esančių klasterių skaičių. Viena iš problemų taikant K-vidurkių algoritmą k reikšmės parinkimas, tam siūlomas alkūnės metodas [237]. Šis algoritmas taip pat derinamas su kitais metodais tikslumui pagerinti [241]. Jis galėtų būti naudojamas toliau segmentuojant esamų segmentavimo algoritmų išvestį.

6.2.3.3. Rankinis segmentavimas

Rankinis vaizdų segmentavimas šiuo metu yra tiksliausias vaizdų segmentavimo metodas [242, 243], tačiau jis užima daug laiko [247]. Programine įranga pagrįsti segmentavimo sprendimai tikslumu nepralenkia rankinio segmentavimo [244, 245], tačiau pusiau automatiniai metodai gali sutaupyti laiko.


Create a 3D-tree representation of point cloud
Set up an empty list of clusters C
Set up a queue of points to check Q
all points in P checked?
no
yes
Take point p from P
Add p to Q
Q has unprocessed points?
yes
Take point p from Q
Search for the set of point neighbors of p in a sphere of radius < d
Add Q to C
Reset Q to empty state
All found points checked?
no
Check if point has already been processed
no
yes
Add point to Q

6.2.1 pav. Euklidinio klasterizavimo algoritmas

6.2.3.4. Žmogaus kūno segmentavimas naudojant neuroninius tinklus

Regionais pagrįsti metodai ištraukia regioną iš paveikslėlio, jį apibūdina ypatybėmis, ir tada jomis mokomas regionų klasifikatorius. Regionai gali būti siūlomi naudojant atrankinę paiešką [256], ypatybės išgaunamos sąsūkos neuroniniu tinklu, o atraminių vektorių mašina nustato objekto klasę. Sąsūkos tinklai susideda tik iš sąsūkos sluoksnių ir gali būti derinami su visiškai sujungtu tinklu globalioms ypatybės surasti, bet prie tinklo prideda daug parametrų [261].

Sąsūkos neuroniniai tinklai išgauna savybes naudojant nelinijines aktyvacijos funkcijas [263, 264]. Segmentavimo neuroniniai tinklai dažnai yra kuriami iš klasifikavimo neuroninių tinklų. Dažna architektūra yra koduoklės ir dekoduoklės tinklas [266]. Praleidimo jungtys naudojamos norint užpildyti semantinį tarpą tarp koduoklės ir dekoduoklės tinklo dalių [267]. Tačiau jų veiksmingumas ginčijamas, kai kurie tyrimai rodo, kad juos pašalinus gali tik šiek tiek sumažėti segmentavimo tikslumas [272].

Segmentavimo uždaviniams dažnai naudojami koduoklės-dekoduoklės architektūros bei U formos tinklai. Abiem atvejais pirmiausia tinklas suspaudžia pradinį vaizdą į tam tikrą savybių vaizdą, o po to iš jo atkuria segmentuotą vaizdą. Suspaudžiančioji dalis, koduoklė, dažnai naudojama klasifikuoti iš esamų tinklų. Vienas pirmųjų sėkmingai taikytų tinklų buvo „AlexNet“ [275]. Jis įkvėpė daug kitų neuroninių tinklų, pvz., „VGG-16“ [27]. Tokių tinklų didžiausias trūkumas yra mažas tikslumas parametrų skaičiui [283]. „VGG-16“ tinklą išplėtus iki U formos buvo sukurtas „SegNet“ tinklas [50], sėkmingai sprendžiantis RGB vaizdų segmentavimo užduotis. Jis naudoja praleidimo jungtis dekoduoklės turimai informacijai papildyti. Modelių dydžių palyginimas pateikiamas 6.2.1 lentelėje. Dvejetainiam segmentavimui dažnai naudojami kelių objektų tipų segmentavimo metodai.

6.2.1 lentelė. Giliojo mašininio mokymosi modelių dydžiai

Modelis	Paskirtis	Parametrai	Modelio failo dydis
AlexNet [275]	RGB klasifikavimas	62M	233 MB
VGG-16 [27]	RGB klasifikavimas	134M	528 MB
SegNet [50]	Semantinis RGB segmentavimas	32M	117 MB
U-Net [267]	RGB dvejetainis segmentavimas	30M	386 MB

6.2.4. Atsitiktiniai miškai

Atsitiktiniai miškai yra sprendimų medžių rinkinys, kurių kiekvienas išmokytas naudojant skirtingą mokymo duomenų pavyzdį. Jie greitai mokosi ir prognozuoja, turi mažai parametrų [313, 314] ir yra taikomi segmentavimui, tačiau tam reikia pateikti jiems aukštesnio lygio savybes [316, 322, 323]. Jie taip pat tinkami segmentuojamo grafo daliai atmesti.

6.2.5. Literatūros apžvalgos apibendrinimas

Literatūros analizė parodė, kad:

- „Azure Kinect“ duomenis sunku panaudoti, nes jie ne stačiakampiai;
- „Kinect“ skeletų sąlajai nėra vieno geriausio sprendimo;
- Gylio žemėlapiams tinka klasikiniai vaizdų apdorojimo algoritmai;
- Aštuntainiai medžiai imlūs atminčiai;
- Dvišalis filtras geriausiai tinka segmentavimui;
- Euklidinis klasterizavimas gali būti adaptuotas segmentavimui;
- Rankinis segmentavimas vis dar yra tiksliausias, bet labai imlus laikui;
- Koduoklės ir dekoduoklės bei U formos architektūros geriausiai tinka segmentavimo uždaviniams;
- Atsitiktiniai miškai gali būti taikomi segmentavimo rezultatui tikslinti.

6.3. Taškų debesų ir skeleto duomenų aprodojimo metodologija

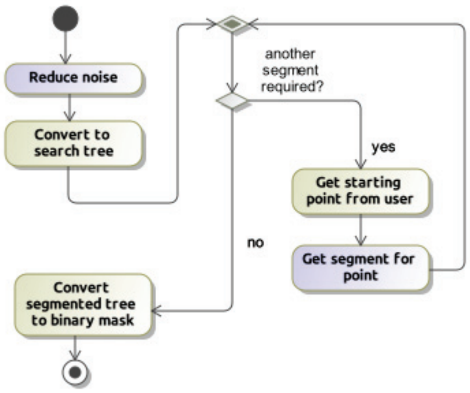
6.3.1. Aukšto lygio proceso apžvalga

Norint sukurti prižiūrimo mašininio mokymosi sprendimą automatiniam segmentavimui, reikia tinkamo duomenų rinkinio. Jo kūrimą reikia iš dalies automatizuoti, kad būtų įmanoma jį paruošti su protingomis laiko sąnaudomis. Dėl to šiame darbe yra siūlomi ir pusiau automatinio segmentavimo algoritmai tokiam duomenų rinkiniui paruošti, ir mašininio mokymosi architektūra, naudojanti šį duomenų rinkinį. Taip pat pateikiami ir algoritmai „Kinect“ skeleto duomenims tikslinti panaudojant kelis jutiklius.

6.3.2. Kryžminės aibių sankirtos metrika

Darbe siūloma kryžminės sankirtos metrika segmentavimo tikslumui įvertinti:

$$C = \frac{|A \cap G|^2}{|A||G|}. \tag{6.7}$$



Reduce noise
Convert to search tree
another segment required?
yes
Get starting point from user
Get segment for point
no
Convert segmented tree to binary mask

6.3.1 pav. Segmentavimo veikla ir jai reikalingi komponentai

Ši metrika pasižymi dviem savybėmis, kurių neturi kitos metrikos (Daiso koeficientas ir Žakardo indeksas): per didelio segmentavimo atveju jos reikšmė parodo, kiek kartų per didelis plotas buvo parinktas, ir garantuoja, kad reikšmė neviršys mažesniojo iš dviejų santykių, kurie yra dauginami. Pastaroji savybė neleidžia gauti didelių įverčių prasto segmentavimo atveju.

6.3.3. Pusiau automatinis segmentavimas

6.3.3.1. Taškų debesų segmentavimas aprėpties dėžutėmis

Tarkime, kad turime gylio kadrų seką. Sprendžiamas uždavinys – kiekvienam kadrui surasti kaukę, kuri parodytų, kurie kadro taškai priklauso žmogaus vaizdui. Sprendimui reikalingi komponentai, analizuojami šiame darbe, parodyti 6.3.1 pav. violetine spalva.

Pasirinkta trimačio dvejetainio medžio struktūra, kurioje taip pat saugoma ir optimizacijoms reikalinga informacija (ar mazgas pašalintas ir ar turi nepašalintų šakų). Medžio dydis atmintyje vienam „Kinect 2“ gylio kadrui ~9,73 MB. Triukšmui mažinti pasirinktas divšalis filtras.

Pusiau automatiniam segmentavimui pagal (6.2) reikalingas parametras m, pateikiamas naudotojo. Jei laikome, kad kiekvienas pikselis gylio vaizde priklauso tik vienam objektui, tai parinkus pikselį galima vienareikšmiškai nusakyti, kurio objekto tai pikselis, ir pagal tai surasti visus jo pikselius, dėl to naudotojui pakanka parinkti bet kurį objekto pikselį kaip parametrą m. Toks parametro parinkimas turi gerą savybę, kad žmogui labai lengva surasti tokį pikselį – jei objektas aiškiai matomas, galima paspausti bet kur ant jo. Segmentavimo metu reikia apibrėžti objekto ribas ir jei randamas taškas, esantis arti šios ribos, jis pridedamas į objektą, tačiau tai reiškia, kad segmentavimo metu keičiasi ir pats objektas, dėl to šį procesą reikia kartoti, iki objektas konverguos. Dar viena užduotis yra surasti tinkamą atstumą, nuo kurio laikoma, kad taškas yra arti objekto ribos, tačiau jį rasti automatiškai sunku, o kartais ir iš viso neįmanoma, priklausomai nuo duomenų. Nors segmentuojant galima naudoti ir euklidiniu klasterizavimu grįstą algoritmą, jo greitaveika nėra gera, kadangi objektas yra apibrėžiamas kaip minisferų rinkinys, dėl to patikrinimas, ar taškas yra arti objekto, turi algoritminį sudėtingumą $O(n)$. Kadangi objektas segmentavimo metu kinta, atmetus tašką dar nereiškia, kad jis iš tiesų nepriklausys objektui. Kiek tikrinimų reikės, priklauso nuo duomenų, bet „Kinect 2“ gylio kadrui reikės tarp 400 tūkst. ir 216 mln. medžio viršūnių patikrinimų.

Euklidinį klasterizavą galima pritaikyti pusiau automatiniam segmentavimui su nedideliais pakeitimais, tačiau jo algoritminis sudėtingumas nėra optimalus. Pagrindinė kliūtis – ilgai trunkantis patikrinimas, ar taškas yra arti objekto. Norint tai išspręsti, reikalingas kitas objekto ribų apribrėžimas. Tą galima pasiekti turint vieną visam objektui bendrą figūrą, kuri nusako jo ribas. Tam tinka aprėpties dėžutė. Tokią dėžutę galima sudaryti nubrėžiant mažiausią galimą stačiakampį gretasienį, apimantį visus objekto taškus. Dėl šios priežasties šiek tiek skiriasi segmentavimo eiga, kadangi patikrinus taškus, kurie priklauso aprėpties dėžutei, reikia atnaujinti ir pačią dėžutę ir vėl kartoti paiešką. Toks paieškos variantas sumažina vidutinį sudėtingumą nuo $\Theta(k \log n)$ iki $\Theta(\log k \log n)$, kur k – objekto dydis, n – taškų debesies dydis, nes objektas didėja greičiau ir reikia mažiau patikrinimų, ar taškas priklauso objektui. Paieškai tada vietoj minisferos dydžio reikalingas kitas hiperparametras – dėžutės jautrumas, kuris nurodo, kiek gali būti nuo dėžutės nutolęs taškas, kad jį būtų galima įtraukti į dėžutę.

Šis algoritmas turi trūkumą, kad jei reikia atlikti paiešką su praplėsta aprėpties dėžute, vėl bus surandami ir grąžinami tie patys taškai, kurie jau buvo surasti. Šiai problemai spręsti reikia turėti papildomą informaciją, kurie taškai jau buvo surasti ankstesnėse iteracijose. Tą galima padaryti pačiame paieškos medyje saugant požymį, ar taškas jau buvo surastas, ir jei buvo, tai jį praleisti. Šis pagerinimas sutaupo darbo, nes nebereikia tikrinti, ar surasti taškai jau buvo pridėti anksčiau. Tai sumažina taškų surinkimo algoritminį sudėtingumą iki tiesinio, konkrečiu pavyzdiniu atveju – nuo 200 mln. iki 20 tūkst. operacijų. Taip pat šis pagerinimas leidžia nebetikrinti, ar taškas jau priklauso kitam segmentui, nes toks taškas tiesiog nebus surastas, jei reikia rasti kelis segmentus. Dėl šios priežasties gali šiek tiek skirtis segmentavimo rezultatas.

Po šio pagerinimo lieka dar viena problema – jeigu visa šaka jau pažymėta kaip susegmentuota, tai ji bus tikrinama be reikalo. Darbe siūloma pridėti dar vieną požymį kiekvienai medžio viršūnei, kuris parodo, ar ši viršūnė dar turi neišsegmentuotų vaikų, ir jei neturi, tai nebetęsti paieškos gilyn.

Dar viena problema su aprėpties dėžutėmis grįstu segmentavimo algoritmu yra lėtas dėžutės padidėjimas. Kiekvienoje iteracijoje dėžutė gali padidėti tik per nurodytą algoritmo jautrumo parametrą. Siūloma trečia optimizacija – plėsti dėžutę ne po iteracijos, bet po kiekvieno surasto taško. Tai leistų dėžutei padidėti greičiau ir dėl to reikėtų mažiau paieškų medyje. Visas šias optimizacijas galima sujungti ir naudoti kartu, pirmos dvi gali būti taikomos ir euklidinei paieškai.

6.3.3.2. Rankinis segmentavimas

Segmentavimą galima atlikti ir vien tik rankiniu būdu, suteikiant galimybę naudoti pelės žymeklį kaip teptuką norimoms zonoms nupiešti. Taip pat naudotojas gali ir nutrinti blogai pažymėtą segmentą ar atšaukti savo veiksmus. Rankinį ir pusiau automatinį segmentavimą galima naudoti kartu bendrame segmentavimo procese. Pusiau automatinis sprendimas darbą atliks greičiau, tačiau ne su visais duomenimis yra tikslus, dėl to tiems atvejams, kai rezultatas nėra teisingas, naudotojas gali pakoreguoti gautą dvejetainę kaukę rankiniu būdu. Taip galima greičiau susegmentuoti duomenis nei rankiniais metodais neprarandant tikslumo.

Kadangi paieškos medžiai naudoja daug atminties (10 sekundžių, 300 kadrų įrašas užima ~3,4 GB atmintyje), taškų debesų rinkiniai apdorojami dalimis. Į atmintį įkraunama dalis informacijos, ji susegmentuojama ir išsaugoma į diską. Tada įkraunama kita dalis. Baigus apdoroti visą įrašą, gautos kaukės sujungiamos į vieną failą.

6.3.3.3. Per mažo segmentavimo mažinimas naudojant rekursinį 2-vidurkių padalijimo algoritmą su atsitiktinio miško klasifikatoriumi padalijimui priimti

Aprėpties dėžutėmis grįstas algoritmas turi tendenciją atlikti per mažą segmentavimą. Šią problemą galima spręsti padalijant rezultatą į dvi dalis ir vieną iš jų atmetant. Padalijimui siūlomas rekursinis 2-vidurkių padalijimo algoritmas su atsitiktinio miško klasifikatoriumi padalijimui priimti, kuris vis dalija segmentą į dvi dalis, kol po padalijimo suprastėja tikslumas. Vienas centroidas yra fiksuotas (tai naudotojo pasirinktas taškas), kitas – judantis. Antrasis centroidas kilnojamas, kol konverguoja. Abiem gautiems klasteriams skaičiuojamos tokios metrikos:

- Vidutiniai atstumai tarp visų kombinacijų tarp abiejų centroidų ir trijų klasterių (pradinio ir abiejų, gautų po padalijimo);
- Abiejų klasterių dydžiai.

Įvertinimas, kada nustoti dalinti, yra atliekamas atsitiktinio miško metodu. Buvo sužymėtos kaukės duomenų rinkiniui, tada parenkamas vienas iš kaukės taškų kaip pirmasis centroidas, bandoma dalinti ir tikrinama, ar pagerėjo tikslumas, t.y. tai, ką ir turėtų prognozuoti atsitiktinis miškas. Šie duomenys naudojami jam mokyti. Klasifikatorius naudoja 9 medžių rinkinį. Klasifikavimo tikslumas buvo 95%.

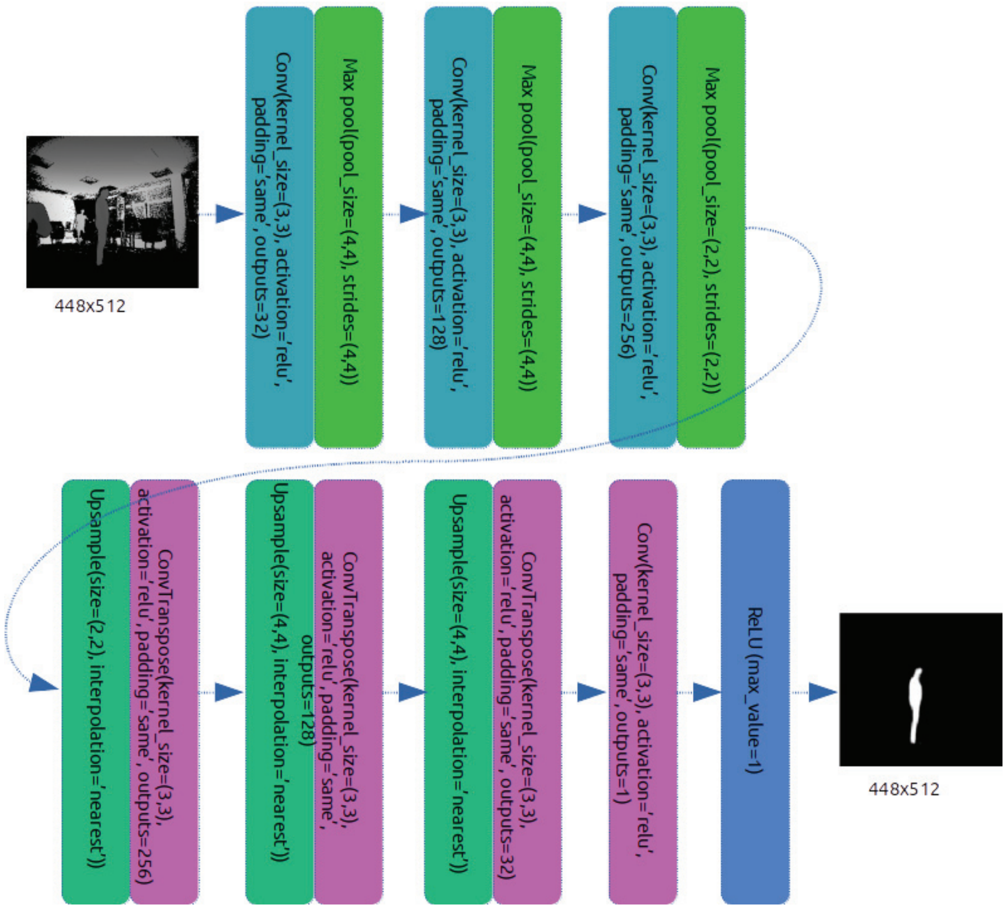
Taikant pasiūlytą metodą bus laimima tikslumo, bet pralaimima greitaveikos, kadangi reikės skaičiuoti papildomas metrikas, vykdyti 2 vidurkių dalijimus bei naudoti atsitiktinį mišką.

6.3.4. Automatinis segmentavimas naudojant „Agrast-6“ architektūrą

„SegNet“ neuroninis tinklas yra vienas populiariausių segmentavimo metodų, tačiau yra didelis ir lėtas. Jis naudojamas kelioms klasėms segmentuoti, dėl to turi saugoti visų jų savybes. Dėl to paprastesnė architektūra turėtų galėti spręsti žmogaus dvejetainio segmentavimo problemą.

Siūloma „Agrast-6“ architektūra yra sutrumpintas ir sumažintas „SegNet“ variantas, tačiau remiasi tomis pačiomis idėjomis. Naudojama koduoklės ir dekoduoklės architektūra, bet be praleidimo jungčių. Koduoklė sumažinta tokiais principais:

- Trys sluoksnių blokai vietoj penkių;
- Vienas sąsūkos sluoksnis vietoj dviejų arba trijų bloke;
- Mažesni sąsūkos sluoksnių gyliai;
- Staigesnis matmenų mažinimas išrenkant maksimalią reikšmę.


448x512
Conv(kernel_size=(3,3), activation='relu', padding='same', outputs=32)
Max pool(pool_size=(4,4), strides=(4,4))
Conv(kernel_size=(3,3), activation='relu', padding='same', outputs=128)
Max pool(pool_size=(4,4), strides=(4,4))
Conv(kernel_size=(3,3), activation='relu', padding='same', outputs=256)
Max pool(pool_size=(2,2), strides=(2,2))
Upsample(size=(2,2), interpolation='nearest'))
ConvTranspose(kernel_size=(3,3), activation='relu', padding='same', outputs=256)
Upsample(size=(4,4), interpolation='nearest'))
ConvTranspose(kernel_size=(3,3), activation='relu', padding='same', outputs=128)
Upsample(size=(4,4), interpolation='nearest'))
ConvTranspose(kernel_size=(3,3), activation='relu', padding='same', outputs=32)
Conv(kernel_size=(3,3), activation='relu', padding='same', outputs=1)
ReLU (max_value=1)
448x512

6.3.2 pav. Siūlomo „Agrast-6“ modelio architektūra

Tinklo architektūra parodyta 6.3.2 pav. Mokomų parametrų skaičius sumažėjo nuo 32 mln. iki 1,25 mln.

Tinklas išmokytas panaudojant jau minėtus duomenų rinkinius. 80% duomenų skirta mokyti, 20% – testuoti. Duomenys padalinti pagal videosekas. Modelio hiperparametrai pateikiami 6.3.1 lentelėje.

6.3.5. Keleto kamerų skeletų transformacija ir sąlaja

6.3.5.1. Skeletų transformacija

Skeletų transformacija atliekama dviem žingsniais:

- Pasukti jutiklio koordinačių sistemą taip, kad xOz plokštuma sutaptų su grindų plokštuma;
- Pritaikyti posūkio ir perkėlimo transformacijas, kad sutaptų jutiklių koordinačių sistemos.

6.3.1 lentelė. „Agrast-6“ modelio hiperparametrų reikšmės

Hiperparametras	Reikšmė
Sąsūkos sluoksnio branduolio dydis	3×3
Sąsūkos sluoksnio aktyvacijos funkcija	Lygintuvas
Maksimalios reikšmės išrinkimo dydis	4×4, 2×2 paskutiniam sluoksniui
Optimizatoirus	Adamo
Optimizatoriaus mokymosi greitis	0,0001
Nuostolio funkcija	Dvejetainė kryžminė entropija

Tai atliekama naudojant šias formules:

$$B_{t1} = B_p T_{p1} - [0, D_p, 0], \tag{6.8}$$

$$\forall q \in B_{t1} : B_{t2} = [\sqrt{x_q^2 + y_q^2} \cos(\varphi_q - \varphi_{r2}), \sqrt{x_q^2 + y_q^2} \sin(\varphi_q - \varphi_{r2})], \tag{6.9}$$

$$B_1 = B_{t2} + T_{p2}, \tag{6.10}$$

čia B_p yra jutiklio p skeleto taškų koordinatės, $T_p 1$ – perkėlimo vektorius, lygus tų vektorių skirtumui tarp jutiklių koordinačių sistemų, D_p – jutiklio p aukštis virš žemės, $\varphi_q - \varphi_{r2}$ – posūkis tarp jutiklių koordinačių sistemų, nustatomas pagal bendro taško matymo kampą. Siūlomas skeletų transformacijos algoritmas nėra iteracinis, dėl to jo vykdymo laikas yra fiksuotas, visi skaičiavimai atliekami arba vieną kartą, arba vieną kartą kiekvienam taškui. Kita vertus, nereikalingas objektas žymeklis, dėl to šis algoritmas turi šio metodo privalumų nenaudojant žymeklio.

6.3.5.2. Skeletų sąlaja

Darbe analizuojami literatūroje siūlomi sprendimai:

- Koordinačių vidurkis bei mediana [105];
- Mažo patikimumo reikšmių atmetimas [99].

6.4. Eksperimentiniai rezultatai

6.4.1. Duomenų rinkiniai

Šio darbo tyrimams naudojami du duomenų rinkiniai, surinkti naudojant tris „Kinect 2“ įrenginius. Pirmasis duomenų rinkinys, toliau darbe vadinamas „sudėtinguoju“, surinktas iš 7 žmonių 30 skirtingų pozicijų. Jame yra 674 gylio vaizdo įrašai (193 tūkst. kadrų) iš trijų kampų (priekio, nugaros ir šono). Antrasis duomenų rinkinys yra 40 žmonių, kurie stovi arba sėdi ant kėdės, – 266 įrašai (69 tūkst. kadrų) iš trijų kampų (prieko, 120 ir 240 laipsnių). Kiekvienas įrašas yra maždaug 10 sek. trukmės, ir jame žmogus juda minimaliai. Pozos detalizuotos priede C.

6.4.2. Pusiau automatinis segmentavimas

6.4.2.1. Eksperimentinė triukšmo mažinimo analizė

Dvišalis filtras segmentavimo uždaviniams visais atvejais veikia tiksliau, nei Gauso ar medianos filtrai. Gauso filtras pablogina tikslumą, palyginti su jokio filt-

6.4.1 lentelė. Euklidinės paieškos ir aprėpties dėžučių segmentavimo rezultatų palyginimas

	Euklidinė paieška (atspirties taškas)	Aprėpties dėžutės
Apdorojimo laikas	2326 ms	6,33 ms
Aplankytų medžio viršūnių skaičius	194 mln.	0,311 mln.
Paprastojo duomenų rinkinio apdorojimo tikslumas	79,8%	82,1%
Sudėtingojo duomenų rinkinio apdorojimo tikslumas	48,1%	33,4%
Tikėtinas bendras duomenų rinkinio paruošimo laikas	28 val.	25 val.

ro netaikymu. Kita vertus, dvišalis filtras pritaikomas dvigubai lėčiau nei mediana ir 4,5 karto lėčiau nei Gauso.

6.4.2.2. Eksperimentinė taškų debesies apdorojimo analizė

Sukurtų segmentavimo algoritmų greitaveikai matuoti naudojamas pilnas gylio kadro segmentavimas. Pasirinkta 1000 atsitiktinių vaizdų ir skaičiuojama jų apdorojimo trukmė. Aprėpties dėžučių algoritmo optimizacija su segmentuotų taškų pažymėjimu pagerino algoritmo greitaveiką 23,4 karto. Automatinio dėžutės didinimo optimizacija pagerino greitaveiką 11,7 karto. Išsegmentuotų šakų praleidimo optimizacija pridėjo 13,5% greitaveikos. Sujungus visas optimizacijas gaunamas 83 kartų pagreitėjimas. Euklidinės paieškos algoritmas veikė net 367 kartus lėčiau, tačiau pritaikius optimizacijas pagreitėjo 4,2 karto.

Paprastam duomenų rinkiniui euklidinės paieškos ir aprėpties dėžučių algoritmai veikė panašiu tikslumu (80% ir 82%), tačiau sudėtingam duomenų rinkiniui euklidinė paieška buvo daug tikslesnė (48% ir 33%).

Abiejų algoritmų tikslumas priklauso nuo kameros kampo – kuo didesnis žmogaus plotas matomas, tuo geresnis tikslumas. Taip pat pastebėta, kad aprėpties dėžučių algoritmas yra mažiau stabilus – gaunamas arba labai geras, arba labai blogas tikslumas. Euklidinės paieškos daromos klaidos buvo tiek per didelis, tiek per mažas segmentavimas, tačiau per mažo segmentavimo klaidos buvo daug mažesnės. Aprėpties dėžučių algoritmas turėjo daugiau problemų dėl per didelio ir mažiau problemų dėl per mažo segmentavimo. Abu algoritmai išsegmentavo daugiau nei 80% paprasto duomenų rinkinio įrašų su didesniu nei 90% tikslumu. Sudėtingam rinkiniui 80% duomenų išsegmentuoti daugiau nei 15% tikslumu, tačiau euklidinei paieškai kreivė krenta lėčiau. Abu algoritmai šiek tiek tiksliau segmentavo moteris nei vyrus.

Abu duomenų rinkiniai buvo susegmentuoti dviejų žmonių per 25 žmogvalandes. Segmentuojant buvo naudojamas aprėpties dėžučių algoritmas, euklidinė paieška būtų pridėjusi papildomas 3 valandas. Jei segmentavimas būtų atliekamas tik rankiniu būdu, būtų užtrukęs 1455 valandas.

6.4.3. Per mažo segmentavimo mažinimo naudojant rekursinį 2-vidurkių padalijimo algoritmą su atsitiktinio miško klasifikatoriumi padalijimui priimti korekcijų ekperimentinis įvertinimas

Rekursinis 2-vidurkių padalijimo algoritmas su atsitiktinio miško klasifikatoriumi padalijimui priimti per mažo segmentavimo tikslinimui prailgina skaičiavimų laiką nuo 6,33 ms iki 46,9 ms. Kita vertus, tai vis tiek daug greičiau nei neoptimizuotas euklidinės paieškos algoritmas. Didžiausią laiko dalį užima taškų, kurie buvo be reikalo pažymėti kaip surasti, būsenos atkūrimas bei pačių metrikų skaičiavimas. Paprastam duomenų rinkiniui ši korekcija turėjo mažai įtakos, nes buvo mažai variantų, kur algoritmas atlieka per mažą segmentavimą. Pastebima ir neteisingų atmetimų, kas atspindi 95% klasifikatoriaus tikslumą (ne 100%), tad tikslumo pagerinimas nevertas sulėtėjimo. Kita vertus, pastebėtas ryškus sudėtingo duomenų rinkinio tikslumo pagerėjimas – blogiausių variantų pakilo nuo 9% iki 34%, bendrai viso rinkinio – nuo 33% iki 55%. Smarkiai sumažėjo 20% tikslumo nesiekiančių kadrų. Toks pagerėjimas smarkiai sumažina rankinio trynimo, nes per mažo segmentavimo plotas blogiausiais atvejais sumažėjo nuo 11,1 karto per didelio iki 2,9 karto per didelio. Atlikus kokybinę klaidų analizę matosi, kad toliau tikslumo didinti nebeleidžia pasirinktas 2 vidurkių algoritmas. Kita vertus, ir dabartinis variantas smarkiai pagerina blogiausio tikslumo rezultatus. Įvertinta, kad, pritaikius šią optimizaciją, segmentavimo laikas galėtų būti sumažintas apie 7,5 val (nuo 25 iki 17,5 val). Tačiau jei visi duomenys būtų iš paprasto duomenų rinkinio, tokio pagerėjimo nebūtų.

6.4.4. „Agrast-6“ mokymas ir įvertinimas

„Agrast-6“ neuroninis tinklas per 4–9 mokymosi epochas pagerino nuostolio funkcijos reikšmę nuo 0,043 iki 0,0325. Testinio duomenų rinkinio rezultatai rodo, kad tinklas nepersimoko ir rezultatai kiekvieną epochą po truputį gerėja. Jau pirmoje epochoje po 400 paveikslėlių testiniame kadre matomas neryškus žmogaus siluetas, po 64,8 tūkst. paveikslėlių tikslumas geresnis, tik galva lieka neryški. Kiekviena epochą galvos tikslumas vis gerėja, problema lieka tik pėdos, kurių tinklas po 9 epochų mokymosi vis dar neranda. Atlikus paveikslėlių apdorojimo kiekviename sluoksnyje analizę rasta, kad kad tinklo koduoklė turi aktyvacijas tik dalyje sluoksnių, dėl to tinklas galėtų būti dar labiau sumažintas. Dekoduoklėje matoma daug skirtingų žmogaus silueto interpretacijų, iš kurių matosi, kad tinklas teisingai išmoksta žmogaus silueto savybes. Dėl to „Agrast-6“, žvelgiant kokybiškai, išmoko atpažinti žmogaus siluetą, ir ši supaprastinta architektūra sprendžia segmentavimo problemą.

Atlikta neuroninio tinklo greitaveikos analizė. Nustatyta, kad prognozavimo laikas yra 166 ms ir yra stabilus. „SegNet“ tinklas su mažesniais paveikslėliais ir silpnesne vaizdo plokšte dirbo 443 ms, tad „Agrast-6“ greitis yra geresnis. „SegNet“ ir „Agrast-6“ palyginimas pateikiamas 6.4.2 lentelėje.

„Agrast-6“ pasiekė 82% tikslumą su paprastu duomenų rinkiniu ir 89% tikslumą su sudėtingu, galimai dėl to, kad sudėtingas rinkinys yra 3 kartus didesnis, bet tai vis tiek rodo, kad tinklas išmoko net ir sudėtingose pozose esantį žmogaus kūną. Geresnis paprasto duomenų rinkinio tikslumas pasiektas stovimose pozicijose ir iš priekinės

6.4.2 lentelė. „SegNet“ ir siūlomo „Agrast-6“ neuroninių tinklų palyginimas

	SegNet	Agrast-6
Modelio dydis diske	117 MB	15,4 MB
Parametrų kiekis	32 mln.	1,25 mln.
Duomenų apdorojimo trukmė	69 ms @ 340x480	34 ms @ 448x512
Vidutinė kryžminė aibių sankirta	89,5%	85,4%

kameros. Kameros matomas plotas turėjo įtakos ir sudėtingam duomenų rinkiniui – geriausias tikslumas iš galinės kameros, kuri mato didžiausią kūno plotą, bet skirtumai nuo kitų kamerų nėra dideli. Gauti rezultatai perša išvadą, kad reikėtų daugiau duomenų arba ilgesnio mokymo paprastam duomenų rinkiniui geriau išmokti. 92% visų kadrų patenka į 80–100% tikslumo rėžį. „Agrast-6“ tinklas daro tiek per didelio, tiek per mažo segmentavimo klaidų, tačiau retai. Kai žmogus dalinai uždengtas arba dėvi sunkiai gylio kamerai matomas kelnes, aptinkama tik dalis žmogaus kūno. Kartais aptinkama kita į žmogaus kūną panaši figūra.

6.4.5. Skeletų transformacijos ir sąlajos eksperimentinis įvertinimas

Skeletų sąlajos algoritmas buvo įvertintas naudojant 2400 kadrų dydžio duomenų rinkinį su ramiai stovinčiais žmonėmis. Matuojamos sąnarių fliuktuacijos, kurios turėtų būti minimalios esant geram tikslumui, nes žmogus stengiasi nejudėti. Koordinačių vidurkis buvo tikslesnis y ir z ašyse, mediana – x ašyje. Reikšmių vidurkinimas iš buferio nedavė norimo efekto, dėl to geriau naudoti tik vieno kadro informaciją. Mažo patikimumo taškų atmetimas taip pat nepagerino tikslumo. Išmatuotas algoritmo veikimo greitis – 243 µs. Sąnarių koordinačių svyravimai sumažinti nuo 10% iki 2%.

6.5. Išvados

Darbe prieita prie tokių išvadų:

1. Literatūros apžvalga parodė, kad dvejetainio segmentavimo problemai gali būti taikomi ir geometriniai, ir mašininio mokymosi metodai, tačiau mašininiu mokymusi pagrįsti metodai paprastai pasižymi didesniu tikslumu, bet jiems reikia daug duomenų mokant, todėl duomenų rinkinio paruošimas yra viena didžiausių kliūčių išmokant segmentavimo neuroninius tinklus.
2. Tiek euklidiniu klasterizavimu, tiek siūlomomis aprėpties dėžutėmis grįsti algoritmai tinka pusiau automatinio dvejetainio gylio segmentavimo problemai spręsti. Euklidiniu klasterizavimo grįstas segmentavimas yra tikslesnis, ypač sudėtingose scenose (48,1% ir 33,4%), tačiau aprėpties dėžutėmis grįstas segmentavimas veikia iki 367 kartų greičiau (2,33 s ir 6,33 ms). Aprėpties dėžučių sprendimas sumažino žmogaus darbo laiką nuo 1455 valandų iki 25 valandų duomenų rinkiniui paruošti.
3. Rekursinis 2-vidurkių padalijimo algoritmas su atsitiktinio miško klasifikatoriumi padalijimui priimti per mažam segmentavimui koreguoti pagerina segmentavimo tikslumą ten, kur jis iš pradžių buvo mažas (33,4% palyginti su 55,3%), bet yra ne toks našus (6,33 ms ir 40,6 ms). Tai parodo, kad naudojant pasiūly-

mą sumažinamas per mažas segmentavimas. Tačiau tikslumą būtų galima dar pagerinti, jei vietoj siūlomo rekursinio 2-vidurkių padalijimo būtų galima rasti geresnį taškų debesies padalijimo algoritmą.

4. Siūlomi pusiau automatiniai segmentavimo algoritmai labai sumažina laiką, reikalingą giluminių vaizdo sekų dvejetainėms kaukėms pažymėti. Naudojant siūlomą programinės įrangos sprendimą jie leido paruošti kaukes 220 000 vaizdų per 25 valandas, o šiuolaikinius duomenų rinkinius panašiems tikslams ruošti dažnai reikia ištisoms komandoms.
5. Siūloma „Agrast-6“ architektūra parodė, kad mažesnės architektūros gali duoti panašų dvejetainio gylio segmentavimo tikslumą, kaip ir didelės architektūros, skirtos RGB segmentuoti. „Agrast-6“ turi 27 kartus mažiau išmokomų parametrų nei „SegNet“, tačiau vis tiek pasiekia 86% dvejetainio segmentavimo tikslumą. Nepaisant to, analizė parodė, kad siūlomos architektūros koduoklė turi mažai aktyvacijų, todėl turėtų būti įmanoma pasiekti panašų tikslumą sumažinant koduoklę.
6. Siūlomas skeletų sąlajos algoritmas sumažino skeleto duomenų svyravimus nuo pradinių 10% naudojant vieną „Kinect“ įrenginį iki 2% naudojant tris įrenginius bei algoritmą. Jis buvo sėkmingai panaudotas tolimesniems tyrimams žmogaus sąnarių apkrovoms fizinės veiklos metu įvertinti.

BIBLIOGRAPHY

1. ROESNER, Franziska; KOHNO, Tadayoshi; MOLNAR, David. Security and privacy for augmented reality systems. *Communications of the ACM*. 2014, vol. 57, no. 4, pp. 88–96.
2. ZHANG, Qing; FU, Bo; YE, Mao; YANG, Ruigang. Quality dynamic human body modeling using a single low-cost depth camera. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 676–683.
3. HU, Tianyu et al. Development and performance evaluation of a very low-cost UAV-LiDAR system for forestry applications. *Remote Sensing*. 2020, vol. 13, no. 1, p. 77.
4. CHEN, Lin-Zhuo et al. Spatial information guided convolution for real-time RGBD semantic segmentation. *IEEE Transactions on Image Processing*. 2021, vol. 30, pp. 2313–2324.
5. MRÁČEK, Štěpán et al. 3D face recognition on low-cost depth sensors. In: *2014 International Conference of the Biometrics Special Interest Group (BIOSIG)*. 2014, pp. 1–4.
6. CIPPITELLI, E.; FIORANELLI, F.; GAMBI, E.; SPINSANTE, S. Radar and RGB-depth sensors for fall detection: A review. *IEEE Sensors Journal*. 2017, vol. 17, no. 12, pp. 3585–3604.
7. KURILLO, G.; CHEN, A.; BAJCSY, R.; HAN, J. J. Evaluation of upper extremity reachable workspace using Kinect camera. *Technology and Health Care*. 2013, vol. 21, no. 6, pp. 641–656.
8. CHEN, Chen; LIU, Kui; JAFARI, Roozbeh; KEHTARNAVAZ, Nasser. Home-based senior fitness test measurement system using collaborative inertial and depth sensors. In: *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. 2014, pp. 4135–4138.
9. RYSELIS, K. et al. Multiple Kinect based system to monitor and analyze key performance indicators of physical training. *Human-centric Computing and Information Sciences*. 2020, vol. 10, no. 1.
10. OFLI, F. et al. Design and evaluation of an interactive exercise coaching system for older adults: Lessons learned. *IEEE Journal of Biomedical and Health Informatics*. 2016, vol. 20, no. 1, pp. 201–212.
11. PATALAS□MALISZEWSKA, J.; HALIKOWSKI, D.; DAMAŠEVIČIUS, R. An automated recognition of work activity in industrial manufacturing using convolutional neural networks. *Electronics (Switzerland)*. 2021, vol. 10, no. 23.
12. TADIC, V. et al. Perspectives of RealSense and ZED Depth Sensors for Robotic Vision Applications. *Machines*. 2022, vol. 10, no. 3.
13. LONG, N. et al. Unifying obstacle detection, recognition, and fusion based on millimeter wave radar and RGB-depth sensors for the visually impaired. *Review of Scientific Instruments*. 2019, vol. 90, no. 4.
14. CAMALAN, S. et al. Gender detection using 3d anthropometric measurements by kinect. *Metrology and Measurement Systems*. 2018, vol. 25, no. 2, pp. 253–267.
15. KULIKAJEVAS, A.; MASKELIUNAS, R.; DAMASEVICIUS, R.; SCHERER, R. Humannet-a two-tiered deep neural network architecture for self-occluding humanoid pose reconstruction. *Sensors*. 2021, vol. 21, no. 12.

16. CARMO VILAS-BOAS, Maria do et al. Full-body motion assessment: Concurrent validation of two body tracking depth sensors versus a gold standard system during gait. *Journal of biomechanics*. 2019, vol. 87, pp. 189–196.
17. MA, Yulin et al. Image encryption scheme based on alternate quantum walks and discrete cosine transform. *Opt. Express*. 2021, vol. 29, no. 18, pp. 28338–28351. Available from doi: `10.1364/OE.431945`.
18. KHANDAY, N. Y.; SOFI, S. A. Taxonomy, state-of-the-art, challenges and applications of visual understanding: A review. *Computer Science Review*. 2021, vol. 40.
19. GARCIA-GARCIA, A. et al. A survey on deep learning techniques for image and video semantic segmentation. *Applied Soft Computing Journal*. 2018, vol. 70, pp. 41–65.
20. ULKU, I.; AKAGÜNDÜZ, E. A Survey on Deep Learning-based Architectures for Semantic Segmentation on 2D Images. *Applied Artificial Intelligence*. 2022.
21. FU, Kun et al. WSF-NET: Weakly supervised feature-fusion network for binary segmentation in remote sensing image. *Remote Sensing*. 2018, vol. 10, no. 12, p. 1970.
22. BARROWCLOUGH, Oliver JD; MUNTINGH, Georg; NAINAMALAI, Varatharajan; STANGEBY, Ivar. Binary segmentation of medical images using implicit spline representations and deep learning. *Computer Aided Geometric Design*. 2021, vol. 85, p. 101972.
23. HU, Yuan-Ting; HUANG, Jia-Bin; SCHWING, Alexander. Maskrnn: Instance level video object segmentation. *Advances in neural information processing systems*. 2017, vol. 30.
24. SHAZEER, Noam et al. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*. 2017.
25. HUANG, Yanping et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*. 2019, vol. 32.
26. WANG, Xiaogang et al. Deep learning in object recognition, detection, and segmentation. *Foundations and Trends® in Signal Processing*. 2016, vol. 8, no. 4, pp. 217–382.
27. SIMONYAN, Karen; ZISSERMAN, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. 2014.
28. ZHU, Zhen et al. Asymmetric non-local neural networks for semantic segmentation. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 593–602.
29. PASZKE, Adam; CHAURASIA, Abhishek; KIM, Sangpil; CULURCIELLO, Eugenio. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*. 2016.
30. NOH, Hyeonwoo; HONG, Seunghoon; HAN, Bohyung. Learning deconvolution network for semantic segmentation. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1520–1528.
31. EVERINGHAM, Mark et al. The pascal visual object classes (voc) challenge. *International journal of computer vision*. 2010, vol. 88, no. 2, pp. 303–338.
32. XU, Chenfeng et al. Squeezesegv3: Spatially-adaptive convolution for efficient point-cloud segmentation. In: *European Conference on Computer Vision*. 2020, pp. 1–19.
33. CHANG, Angel X et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*. 2015.

34. OVED, Dan; ZHU, Tyler. *BodyPix: Real-time Person Segmentation in the Browser with TensorFlow.js* [online]. 2019 [visited on 2022-07-09]. Available from: `https://blog.tensorflow.org/2019/11/updated-bodypix-2.html`.

35. LEHMENT, Nicolas; KAISER, Moritz; RIGOLL, Gerhard. Using segmented 3D point clouds for accurate likelihood approximation in human pose tracking. *International journal of computer vision*. 2013, vol. 101, no. 3, pp. 482–497.

36. HU, Pengpeng; HO, Edmond Shu–Lim; MUNTEANU, Adrian. 3DBodyNet: fast reconstruction of 3D animatable human body shape from a single commodity depth camera. *IEEE Transactions on Multimedia*. 2021, vol. 24, pp. 2139–2149.

37. RYSELIS, Karolis; PETKUS, Tautvydas. Nestandartinių žmogaus kūno pozicijų atpažinimo tikslumo naudojant „Kinect 2.0“ jutiklius tyrimas. *XX tarpuniversitetinės magistrantų ir doktorantų konferencijos „Informacinė visuomenė ir universitetinės studijos“ (IVUS 2015) pranešimų medžiaga ”Informacinės technologijos”*. 2015.

38. PAPADOPOULOS, Georgios Th; AXENOPOULOS, Apostolos; DARAS, Petros. Real-time skeleton-tracking-based human action recognition using kinect data. In: *International Conference on Multimedia Modeling*. 2014, pp. 473–483.

39. NAVA, Armando; GARRIDO, Leonardo; BRENA, Ramon F. Recognizing activities using a kinect skeleton tracking and hidden markov models. In: *2014 13th Mexican International Conference on Artificial Intelligence*. 2014, pp. 82–88.

40. WIEDERHOLD, B; RIVA, G. Balance recovery through virtual stepping exercises using Kinect skeleton tracking: a followup study with chronic stroke patients. *Annual review of cybertherapy and telemedicine 2012: Advanced technologies in the behavioral, social and neurosciences*. 2012, vol. 181, pp. 108–112.

41. ELTOUKHY, Moataz A; KUENZE, Christopher; OH, Jeonghoon; SIGNORILE, Joseph F. Validation of static and dynamic balance assessment using Microsoft Kinect for young and elderly populations. *IEEE journal of biomedical and health informatics*. 2017, vol. 22, no. 1, pp. 147–153.

42. CLARK, Ross A et al. Validity of the Microsoft Kinect for assessment of postural control. *Gait & posture*. 2012, vol. 36, no. 3, pp. 372–377.

43. DEHBANDI, Behdad et al. Using data from the Microsoft Kinect 2 to determine postural stability in healthy subjects: A feasibility trial. *PloS one*. 2017, vol. 12, no. 2, e0170890.

44. GALNA, Brook et al. Accuracy of the Microsoft Kinect sensor for measuring movement in people with Parkinson’s disease. *Gait & posture*. 2014, vol. 39, no. 4, pp. 1062–1068.

45. WANG, Baoliang; CHEN, Zeyu; CHEN, Jing. Gesture recognition by using kinect skeleton tracking system. In: *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*. 2013, vol. 1, pp. 418–422.

46. DESTELLE, François et al. Low-cost accurate skeleton tracking based on fusion of kinect and wearable inertial sensors. In: *2014 22nd European Signal Processing Conference (EUSIPCO)*. 2014, pp. 371–375.

47. LIAO, Yajie et al. Simultaneous calibration: a joint optimization approach for multiple kinect and external cameras. *Sensors*. 2017, vol. 17, no. 7, p. 1491.

48. LI, Saiyi; PATHIRANA, Pubudu N; CAELLI, Terry. Multi-kinect skeleton fusion for physical rehabilitation monitoring. In: *2014 36th annual international conference of the ieee engineering in medicine and biology society*. 2014, pp. 5060–5063.

49. RUSU, Radu Bogdan. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. 2009. PhD thesis. Computer Science department, Technische Universitaet Muenchen, Germany.

50. BADRINARAYANAN, Vijay; KENDALL, Alex; CIPOLLA, Roberto. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*. 2017, vol. 39, no. 12, pp. 2481–2495.

51. *Fastest-selling gaming peripheral* [online] [visited on 2022-06-12]. Available from: `https://www.guinnessworldrecords.com/world-records/fastest-selling-gaming-peripheral`.

52. HACHAJ, Tomasz; OGIELA, Marek R; KOPTYRA, Katarzyna. Effectiveness comparison of Kinect and Kinect 2 for recognition of Oyama karate techniques. In: *2015 18th International Conference on Network-Based Information Systems*. 2015, pp. 332–337.

53. SAMIR, Mohammed; GOLKAR, Ehsan; RAHNI, Ashrani Aizzuddin Abd. Comparison between the Kinect™ V1 and Kinect™ V2 for respiratory motion tracking. In: *2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*. 2015, pp. 150–155.

54. ABREU, João et al. Assessment of microsoft kinect in the monitoring and rehabilitation of stroke patients. In: *World Conference on Information Systems and Technologies*. 2017, pp. 167–174.

55. TÖLGYESSY, Michal; DEKAN, Martin; CHOVANEC, L'uboš; HUBINSKỲ, Peter. Evaluation of the azure Kinect and its comparison to Kinect V1 and Kinect V2. *Sensors*. 2021, vol. 21, no. 2, p. 413.

56. LUO, Junren; ZHANG, Wanpeng; SU, Jiongming; XIANG, Fengtao. Hexagonal convolutional neural networks for hexagonal grids. *IEEE Access*. 2019, vol. 7, pp. 142738–142749.

57. ZHAO, Yunxiang et al. HexCNN: A Framework for Native Hexagonal Convolutional Neural Networks. In: *2020 IEEE International Conference on Data Mining (ICDM)*. 2020, pp. 1424–1429.

58. ALBERT, Justin Amadeus et al. Evaluation of the pose tracking performance of the azure kinect and kinect v2 for gait analysis in comparison with a gold standard: A pilot study. *Sensors*. 2020, vol. 20, no. 18, p. 5104.

59. ADIKARI, Sasadara B; GANEGODA, Naleen C; MEEGAMA, Ravinda GN; WANNIARACHCHI, Indika L. Applicability of a single depth sensor in real-time 3D clothes simulation: augmented reality virtual dressing room using kinect sensor. *Advances in Human-Computer Interaction*. 2020, vol. 2020.

60. CONLY, Christopher; ZHANG, Zhong; ATHITSOS, Vassilis. An integrated RGB-D system for looking up the meaning of signs. In: *Proceedings of the 8th ACM International Conference on PErvasive Technologies Related to Assistive Environments*. 2015, pp. 1–8.

61. MALLICK, Tanwi; DAS, Partha Pratim; MAJUMDAR, Arun Kumar. Characterizations of noise in Kinect depth images: A review. *IEEE Sensors journal*. 2014, vol. 14, no. 6, pp. 1731–1740.

62. CHEN, Li; LIN, Hui; LI, Shutao. Depth image enhancement for Kinect using region growing and bilateral filter. In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. 2012, pp. 3070–3073.

63. FU, Jingjing et al. Kinect-like depth denoising. In: *2012 IEEE international symposium on circuits and systems (ISCAS)*. 2012, pp. 512–515.

64. SHIRES, Luke et al. Enhancing the tracking capabilities of the Microsoft Kinect for stroke rehabilitation. In: *2013 IEEE 2nd International Conference on Serious Games and Applications for Health (SeGAH)*. 2013, pp. 1–8.

65. FANKHAUSER, Péter et al. Kinect v2 for mobile robot navigation: Evaluation and modeling. In: *2015 International Conference on Advanced Robotics (ICAR)*. 2015, pp. 388–394.

66. IBRAHIM, Mostafa Mahmoud et al. Depth map artefacts reduction: A review. *IET Image Processing*. 2020, vol. 14, no. 12, pp. 2630–2644.

67. SWEENEY, Chris; IZATT, Greg; TEDRAKE, Russ. A supervised approach to predicting noise in depth images. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 796–802.

68. KURNIAWAN, Aditya; WARDANI, Kholilatul. Kinect Structural Noise Elimination Technique For ITIS Mobile Robot Data Collector. *International Journal of Engineering & Technology*. 2018, vol. 7, no. 4.27, pp. 1–5.

69. INGALE, Anupama K et al. Real-time 3D reconstruction techniques applied in dynamic scenes: A systematic literature review. *Computer Science Review*. 2021, vol. 39, p. 100338.

70. SHUM, Hubert PH; HO, Edmond SL; JIANG, Yang; TAKAGI, Shu. Real-time posture reconstruction for microsoft kinect. *IEEE transactions on cybernetics*. 2013, vol. 43, no. 5, pp. 1357–1369.

71. HO, Edmond SL et al. Improving posture classification accuracy for depth sensor-based human activity monitoring in smart environments. *Computer Vision and Image Understanding*. 2016, vol. 148, pp. 97–110.

72. KULIKAJEVAS, Audrius; MASKELIUNAS, Rytis; DAMAŠEVIČIUS, Robertas. Detection of sitting posture using hierarchical image composition and deep learning. *PeerJ computer science*. 2021, vol. 7, e442.

73. OTTO, Michael et al. Applicability evaluation of kinect for EAWS ergonomic assessments. *Procedia CIRP*. 2019, vol. 81, pp. 781–784.

74. QIN, Hongxing et al. PointSkelCNN: Deep Learning-Based 3D Human Skeleton Extraction from Point Clouds. In: *Computer Graphics Forum*. 2020, vol. 39, pp. 363–374. No. 7.

75. KULIKAJEVAS, Audrius; MASKELIUNAS, Rytis; DAMAŠEVIČIUS, Robertas. Adversarial 3D Human Pointcloud Completion from Limited Angle Depth Data. *IEEE Sensors Journal*. 2021, vol. 21, no. 24, pp. 27757–27765.

76. MOROZOV, MN; SHUBIN, AA; NAIDENOV, KM; DERBENEV, AA. Physical Bases of a ToF Camera–Based Optical Tracking System for Surgical Instruments. *Bulletin of the Russian Academy of Sciences: Physics*. 2018, vol. 82, no. 12, pp. 1525–1528.

77. SARBOLANDI, Hamed; LEFLOCH, Damien; KOLB, Andreas. Kinect range sensing: Structured-light versus Time-of-Flight Kinect. *Computer vision and image understanding*. 2015, vol. 139, pp. 1–20.

78. ALHWARIN, Faraj; FERREIN, Alexander; SCHOLL, Ingrid. IR stereo kinect: improving depth images by combining structured light with IR stereo. In: *Pacific Rim International Conference on Artificial Intelligence*. 2014, pp. 409–421.

79. BHATEJA, Aditi et al. Depth analysis of kinect v2 sensor in different mediums. *Multimedia Tools and Applications*. 2022, vol. 81, no. 25, pp. 35775–35800.

80. STOMMEL, Martin; BEETZ, Michael; XU, Weiliang. Inpainting of missing values in the Kinect sensor's depth maps based on background estimates. *IEEE Sensors Journal*. 2013, vol. 14, no. 4, pp. 1107–1116.

81. HUANG, Jie; ZHOU, Wengang; LI, Houqiang; LI, Weiping. Sign language recognition using real-sense. In: *2015 IEEE China Summit and International Conference on Signal and Information Processing (ChinaSIP)*. 2015, pp. 166–170.

82. NEUPANE, Chiranjivi; KOIRALA, Anand; WANG, Zhenglin; WALSH, Kerry Brian. Evaluation of depth cameras for use in fruit localization and sizing: Finding a successor to kinect v2. *Agronomy*. 2021, vol. 11, no. 9, p. 1780.

83. YIN, Jingfang; ZHU, Dengming; SHI, Min; WANG, Zhaoqi. Depth maps restoration for human using RealSense. *IEEE Access*. 2019, vol. 7, pp. 112544–112553.

84. HALMETSCHLAGER-FUNEK, Georg; SUCHI, Markus; KAMPEL, Martin; VINCZE, Markus. An empirical evaluation of ten depth cameras: Bias, precision, lateral noise, different lighting conditions and materials, and multiple sensor setups in indoor environments. *IEEE Robotics & Automation Magazine*. 2018, vol. 26, no. 1, pp. 67–77.

85. ELARABY, Ahmed Fawzy; HAMDY, Ayman; REHAN, Mohamed. A kinect-based 3d object detection and recognition system with enhanced depth estimation algorithm. In: *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. 2018, pp. 247–252.

86. ENGEL, Jakob; STÜCKLER, Jörg; CREMERS, Daniel. Large-scale direct SLAM with stereo cameras. In: *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. 2015, pp. 1935–1942.

87. USENKO, Vladyslav; ENGEL, Jakob; STÜCKLER, Jörg; CREMERS, Daniel. Direct visual-inertial odometry with stereo cameras. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1885–1892.

88. MUR-ARTAL, Raul; TARDÓS, Juan D. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*. 2017, vol. 33, no. 5, pp. 1255–1262.

89. WANG, Rui; SCHWORER, Martin; CREMERS, Daniel. Stereo DSO: Large-scale direct sparse visual odometry with stereo cameras. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3903–3911.

90. SAPONARO, Philip; SORENSEN, Scott; RHEIN, Stephen; KAMBHAMETTU, Chandra. Improving calibration of thermal stereo cameras using heated calibration board. In: *2015 IEEE International Conference on Image Processing (ICIP)*. 2015, pp. 4718–4722.

91. FATHI, Habib; BRILAKIS, Ioannis. Multistep explicit stereo camera calibration approach to improve euclidean accuracy of large-scale 3D reconstruction. 2016.

92. GENOVESE, Katia; CHI, Yuxi; PAN, Bing. Stereo-camera calibration for large-scale DIC measurements with active phase targets and planar mirrors. *Optics express*. 2019, vol. 27, no. 6, pp. 9040–9053.

93. GAO, Zeren et al. Stereo camera calibration for large field of view digital image correlation using zoom lens. *Measurement*. 2021, vol. 185, p. 109999.

94. ZOETGNANDÉ, Yannick Wend Kuni; FOUGÈRES, Alain-Jérôme; CORMIER, Geoffroy; DILLENSEGER, Jean-Louis. Robust low resolution thermal stereo camera calibration. In: *Eleventh International Conference on Machine Vision (ICMV 2018)*. 2019, vol. 11041, pp. 353–360.

95. PAN, Liyuan et al. Joint stereo video deblurring, scene flow estimation and moving object segmentation. *IEEE Transactions on Image Processing*. 2019, vol. 29, pp. 1748–1761.

96. OTTONELLI, Simona; SPAGNOLO, Paolo; MAZZEO, Pier Luigi; LEO, Marco. Improved video segmentation with color and depth using a stereo camera. In: *2013 IEEE international conference on industrial technology (ICIT)*. 2013, pp. 1134–1139.

97. KAKEGAWA, Shinji; MATONO, Haruki; KIDO, Hideaki; SHIMA, Takeshi. Road surface segmentation based on vertically local disparity histogram for stereo camera. *International Journal of Intelligent Transportation Systems Research*. 2018, vol. 16, no. 2, pp. 90–97.

98. BESL, Paul J; MCKAY, Neil D. Method for registration of 3-D shapes. In: *Sensor fusion IV: control paradigms and data structures*. 1992, vol. 1611, pp. 586–606.

99. KITSIKIDIS, Alexandros; DIMITROPOULOS, Kosmas; DOUKA, Stella; GRAMMALIDIS, Nikos. Dance analysis using multiple kinect sensors. In: *2014 international conference on computer vision theory and applications (VISAPP)*. 2014, vol. 2, pp. 789–795.

100. SUSANTO, Wandi; ROHRBACH, Marcus; SCHIELE, Bernt. 3D object detection with multiple kinects. In: *European Conference on Computer Vision*. 2012, pp. 93–102.

101. MOHAMMADI, Shahram; GERVEI, Omid. Three Dimensional Posed Face Recognition with an Improved Iterative Closest Point Method. *3D Research*. 2019, vol. 10, no. 3, pp. 1–17.

102. ASTERIADIS, Stylianos et al. Estimating human motion from multiple kinect sensors. In: *Proceedings of the 6th international conference on computer vision/computer graphics collaboration techniques and applications*. 2013, pp. 1–6.

103. KOWALSKI, Marek; NARUNIEC, Jacek; DANILUK, Michal. Livescan3d: A fast and inexpensive 3d data acquisition system for multiple kinect v2 sensors. In: *2015 international conference on 3D vision*. 2015, pp. 318–325.

104. NGUYEN, Manh-Hung; HSIAO, Ching-Chun; CHENG, Wen-Huang; HUANG, Ching-Chun. Practical 3D human skeleton tracking based on multi-view and multi-Kinect fusion. *Multimedia Systems*. 2022, vol. 28, no. 2, pp. 529–552.

105. KAENCHAN, Suttipong; MONGKOLNAM, Pornchai; WATANAPA, Bunthit; SATHIENPONG, Sasipa. Automatic multiple kinect cameras setting for simple walking posture analysis. In: *2013 international computer science and engineering conference (ICSEC)*. 2013, pp. 245–249.

106. BAEK, Seongmin; KIM, Myunggyu. Dance experience system using multiple kinects. *International Journal of Future Computer and Communication*. 2015, vol. 4, no. 1, p. 45.

107. RUCHAY, Alexey N; DOROFEEV, Konstantin A; KOLPAKOV, Vladimir I. Fusion of information from multiple Kinect sensors for 3D object reconstruction. *Компьютерная оптика*. 2018, vol. 42, no. 5, pp. 898–903.

108. VASUDEVAN, Vinita; RAMAKRISHNA, M. A hierarchical singular value decomposition algorithm for low rank matrices. *arXiv preprint arXiv:1710.02812*. 2017.

109. MOON, Sungphill; PARK, Youngbin; KO, Dong Wook; SUH, Il Hong. Multiple kinect sensor fusion for human skeleton tracking using Kalman filtering. *International Journal of Advanced Robotic Systems*. 2016, vol. 13, no. 2, p. 65.

110. MORATO, Carlos; KAIPA, Krishnanand N; ZHAO, Boxuan; GUPTA, Satyandra K. Toward safe human robot collaboration by using multiple kinects based real-time human tracking. *Journal of Computing and Information Science in Engineering*. 2014, vol. 14, no. 1.

111. HAZRA, Sumit; PRATAP, Acharya Aditya; TRIPATHY, Dattatreya; NANDY, Anup. Novel data fusion strategy for human gait analysis using multiple kinect sensors. *Biomedical Signal Processing and Control*. 2021, vol. 67, p. 102512.

112. COUPRIE, Camille; FARABET, Clément; NAJMAN, Laurent; LECUN, Yann. Indoor semantic segmentation using depth information. *arXiv preprint arXiv:1301.3572*. 2013.

113. TE, Gusi; HU, Wei; ZHENG, Amin; GUO, Zongming. RGCNN: Regularized Graph CNN for Point Cloud Segmentation. In: *Proceedings of the 26th ACM international conference on Multimedia*. 2018, pp. 746–754.

114. WANG, Weiyue; NEUMANN, Ulrich. Depth-aware cnn for rgb-d segmentation. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 135–150.

115. JU, Miso et al. A Kinect-based segmentation of touching-pigs for real-time monitoring. *Sensors*. 2018, vol. 18, no. 6, p. 1746.

116. MUSSI, Elisa et al. A novel ear elements segmentation algorithm on depth map images. *Computers in Biology and Medicine*. 2021, vol. 129, p. 104157.

117. LI, Dawei et al. Leaf Segmentation on Dense Plant Point Clouds with Facet Region Growing. *Sensors*. 2018, vol. 18, no. 11, p. 3625.

118. MU, Y; ZHOU, G; WANG, H. Canopy Lidar Point Cloud Data K-Means Clustering Watershed Segmentation Method. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*. 2020, vol. 6.

119. LI, Shifeng; LU, Huchuan; ZHANG, Lei. Arbitrary body segmentation in static images. *Pattern recognition*. 2012, vol. 45, no. 9, pp. 3402–3413.

120. HSIEH, I-Hsuan et al. Outdoor walking guide for the visually-impaired people based on semantic segmentation and depth map. In: *2020 International Conference on Pervasive Artificial Intelligence (ICPAI)*. 2020, pp. 144–147.

121. POUDEL, Rudra PK; LIWICKI, Stephan; CIPOLLA, Roberto. Fast-scnn: Fast semantic segmentation network. *arXiv preprint arXiv:1902.04502*. 2019.

122. KANG, Byeongkeun; LEE, Yeejin; NGUYEN, Truong Q. Depth-adaptive deep neural network for semantic segmentation. *IEEE Transactions on Multimedia*. 2018, vol. 20, no. 9, pp. 2478–2490.

123. KIM, Sangpil; CHI, Hyung-Gun. First-Person View Hand Segmentation of Multi-Modal Hand Activity Video Dataset. *BMVC 2020*. 2020.

124. PALMERO, Cristina et al. Multi-modal RGB–Depth–Thermal Human Body Segmentation. *International Journal of Computer Vision*. 2016, vol. 118, no. 2, pp. 217–239.

125. HUANG, Lei et al. Robust human body segmentation based on part appearance and spatial constraint. *Neurocomputing*. 2013, vol. 118, pp. 191–202.

126. ZHAO, Yang; LIU, Zicheng; YANG, Lu; CHENG, Hong. Combing rgb and depth map features for human activity recognition. In: *Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference*. 2012, pp. 1–4.

127. PENG, Houwen et al. RGBD salient object detection: A benchmark and algorithms. In: *European conference on computer vision*. 2014, pp. 92–109.

128. QI, Xiaojuan et al. 3d graph neural networks for rgbd semantic segmentation. In: *IEEE International Conference on Computer Vision*. 2017, pp. 5199–5208.

129. WANG, Jinghua et al. Learning common and specific features for RGB-D semantic segmentation with deconvolutional networks. In: *European Conference on Computer Vision*. 2016, pp. 664–679.

130. HU, Xinxin; YANG, Kailun; FEI, Lei; WANG, Kaiwei. Acnet: Attention based network to exploit complementary features for rgbd semantic segmentation. In: *2019 IEEE International Conference on Image Processing (ICIP)*. 2019, pp. 1440–1444.

131. QIU, Zhouyan et al. Low-cost mobile mapping system solution for traffic sign segmentation using Azure Kinect. *International Journal of Applied Earth Observation and Geoinformation*. 2022, vol. 112, p. 102895.

132. GHAFFARI, Mina; SOWMYA, Arcot; OLIVER, Ruth. Automated brain tumor segmentation using multimodal brain scans: a survey based on models submitted to the BraTS 2012–2018 challenges. *IEEE reviews in biomedical engineering*. 2019, vol. 13, pp. 156–168.

133. VISSER, Eelke et al. Automatic segmentation of the striatum and globus pallidus using MIST: Multimodal Image Segmentation Tool. *NeuroImage*. 2016, vol. 125, pp. 479–497.

134. BUI, Hieu Minh et al. Using grayscale images for object recognition with convolutional-recursive neural network. In: *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*. 2016, pp. 321–325.

135. HE, Yihui. Estimated depth map helps image classification. *arXiv preprint arXiv:1709.07077*. 2017.

136. FOOLADGAR, Fahimeh; KASAEI, Shohreh. A survey on indoor RGB-D semantic segmentation: from hand-crafted features to deep convolutional neural networks. *Multimedia Tools and Applications*. 2020, vol. 79, no. 7, pp. 4499–4524.

137. MAJDI, Arafa; BAKKAY, Mohamed Chafik; ZAGROUBA, Ezzeddine. 3d modeling of indoor environments using kinect sensor. In: *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013)*. 2013, pp. 67–72.

138. KEPSKI, Michal; KWOLEK, Bogdan. Unobtrusive fall detection at home using kinect sensor. In: *International Conference on Computer Analysis of Images and Patterns*. 2013, pp. 457–464.

139. BENTLEY, Jon Luis. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*. 1975.

140. XIE, Yuxing; TIAN, Jiaojiao; ZHU, Xiao Xiang. Linking points with labels in 3D: A review of point cloud semantic segmentation. *IEEE Geoscience and Remote Sensing Magazine*. 2020, vol. 8, no. 4, pp. 38–59.

141. ZHANG, Jiaying; ZHAO, Xiaoli; CHEN, Zheng; LU, Zhejun. A review of deep learning-based semantic segmentation for point cloud. *IEEE Access*. 2019, vol. 7, pp. 179118–179133.

142. NAJDATAEI, Hannaneh; NIKOLAKOPOULOS, Yiannis; GULISANO, Vincenzo; PAPATRIANTAFILOU, Marina. Continuous and parallel lidar point-cloud clustering. In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 671–684.

143. ZHANG, Ling; ZHU, Zhigang. Unsupervised feature learning for point cloud understanding by contrasting and clustering using graph convolutional neural networks. In: *2019 International Conference on 3D Vision (3DV)*. 2019, pp. 395–404.

144. ZHU, Youlian; HUANG, Cheng. An improved median filtering algorithm for image noise reduction. *Physics Procedia*. 2012, vol. 25, pp. 609–616.

145. HACKEL, Timo et al. Semantic3d. net: A new large-scale point cloud classification benchmark. *arXiv preprint arXiv:1704.03847*. 2017.

146. UY, Mikaela Angelina et al. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1588–1597.

147. BEN-SHABAT, Yizhak; LINDENBAUM, Michael; FISCHER, Anath. 3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks. *IEEE Robotics and Automation Letters*. 2018, vol. 3, no. 4, pp. 3145–3152.

148. HOARE, Charles AR. Quicksort. *The computer journal*. 1962, vol. 5, no. 1, pp. 10–16.

149. KIRSCHENHOFER, Peter; PRODINGER, Helmut; MARTINEZ, Conrado. Analysis of Hoare's FIND algorithm with Median-of-three partition. *Random Structures & Algorithms*. 1997, vol. 10, no. 1-2, pp. 143–156.

150. BROWN, Russell A. Building a balanced kd tree in o (kn log n) time. *arXiv preprint arXiv:1410.5420*. 2014.

151. HU, Linjia; NOOSHABADI, Saeid; AHMADI, Majid. Massively parallel KD-tree construction and nearest neighbor search algorithms. In: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2015, pp. 2752–2755.

152. WEHR, David; RADKOWSKI, Rafael. Parallel kd-tree construction on the GPU with an adaptive split and sort strategy. *International Journal of Parallel Programming*. 2018, vol. 46, no. 6, pp. 1139–1156.

153. HU, Linjia; NOOSHABADI, Saeid. High-dimensional image descriptor matching using highly parallel KD-tree construction and approximate nearest neighbor search. *Journal of Parallel and Distributed Computing*. 2019, vol. 132, pp. 127–140.

154. CAI, Yixi; XU, Wei; ZHANG, Fu. ikd-Tree: An incremental KD tree for robotic applications. *arXiv preprint arXiv:2102.10808*. 2021.

155. DOORNIK, J.A.; HANSEN, H. *Octree Encoding: A New Technique For The Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. 1980. Tech. rep. Rensseriaer Polytechnic Institute.

156. VO, Anh-Vu; TRUONG-HONG, Linh; LAEFER, Debra F; BERTOLOTTO, Michela. Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2015, vol. 104, pp. 88–100.

157. SU, Yun-Ting; BETHEL, James; HU, Shuowen. Octree-based segmentation for terrestrial LiDAR point cloud data in industrial applications. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2016, vol. 113, pp. 59–74.

158. GUPTA, Kunal et al. Octree Representation Improves Data Fidelity of Cardiac CT Images and Convolutional Neural Network Semantic Segmentation of Left Atrial and Ventricular Chambers. *Radiology: Artificial Intelligence*. 2021, vol. 3, no. 6.

159. XIANG, Binbin; TU, Jingmin; YAO, Jian; LI, Li. A novel octree-based 3-D fully convolutional neural network for point cloud classification in road environment. *IEEE Transactions on Geoscience and Remote Sensing*. 2019, vol. 57, no. 10, pp. 7799–7818.

160. MUZAHID, AAM et al. 3D Object classification using a volumetric deep neural network: An efficient Octree Guided Auxiliary Learning approach. *IEEE Access*. 2020, vol. 8, pp. 23802–23816.

161. STOJANOVIC, Vladeta; TRAPP, Matthias; RICHTER, Rico; DÖLLNER, Jürgen. Service-oriented semantic enrichment of indoor point clouds using octree-based multiview classification. *Graphical Models*. 2019, vol. 105, p. 101039.

162. ZENG, Ming; ZHAO, Fukai; ZHENG, Jiaxiang; LIU, Xinguo. Octree-based fusion for realtime 3D reconstruction. *Graphical Models*. 2013, vol. 75, no. 3, pp. 126–136.

163. STRÖTER, Daniel; MUELLER-ROEMER, Johannes S; STORK, André; FELLNER, Dieter W. OLBVH: octree linear bounding volume hierarchy for volumetric meshes. *The Visual Computer*. 2020, vol. 36, no. 10, pp. 2327–2340.

164. SAFTLY, Waad; BAES, Maarten; CAMPS, Peter. Hierarchical octree and kd tree grids for 3D radiative transfer simulations. *Astronomy & Astrophysics*. 2014, vol. 561, A77.

165. VERMA, Rohit; ALI, Jahid. A comparative study of various types of image noise and efficient noise removal techniques. *International Journal of advanced research in computer science and software engineering*. 2013, vol. 3, no. 10.

166. ERKAN, Uğur; GÖKREM, Levent; ENGINOĞLU, Serdar. Different applied median filter in salt and pepper noise. *Computers & Electrical Engineering*. 2018, vol. 70, pp. 789–798.

167. HSIEH, Mu-Hsien; CHENG, Fan-Chieh; SHIE, Mon-Chau; RUAN, Shanq-Jang. Fast and efficient median filter for removing 1–99% levels of salt-and-pepper noise in images. *Engineering Applications of Artificial Intelligence*. 2013, vol. 26, no. 4, pp. 1333–1338.

168. JASSIM, Firas Ajil; ALTAANI, Fawzi H. Hybridization of Otsu method and median filter for color image segmentation. *arXiv preprint arXiv:1305.1052*. 2013.

169. ZHANG, Peixuan; LI, Fang. A new adaptive weighted mean filter for removing salt-and-pepper noise. *IEEE signal processing letters*. 2014, vol. 21, no. 10, pp. 1280–1283.

170. SHAH, Anwar et al. Comparative analysis of median filter and its variants for removal of impulse noise from gray scale images. *Journal of King Saud University-Computer and Information Sciences*. 2020.

171. SHETTI, Pravin P; PATIL, AP. Performance comparison of mean, median and wiener filter in MRI image de-noising. *International Journal for Research Trends and Innovation*. 2017, vol. 2, no. 371-375.

172. CASTANEDA, Raul; GARCIA-SUCERQUIA, Jorge; DOBLAS, Ana. Speckle noise reduction in coherent imaging systems via hybrid median–mean filter. *Optical Engineering*. 2021, vol. 60, no. 12, p. 123107.

173. VIKRAMATHITHAN, AC; BHAT, Sourabh V; SHASHIKUMAR, DR. Denoising High Density Impulse Noise using Duo-Median Filter for Mammogram Images. In: *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*. 2020, pp. 610–613.

174. HULIK, Rostislav; SPANEL, Michal; SMRZ, Pavel; MATERNA, Zdenek. Continuous plane detection in point-cloud data based on 3D Hough Transform. *Journal of visual communication and image representation*. 2014, vol. 25, no. 1, pp. 86–97.

175. YUN, Ting et al. Individual tree crown segmentation from airborne LiDAR data using a novel Gaussian filter and energy function minimization-based approach. *Remote Sensing of Environment*. 2021, vol. 256, p. 112307.

176. ANDRIA, G et al. Linear filtering of 2-D wavelet coefficients for denoising ultrasound medical images. *Measurement*. 2012, vol. 45, no. 7, pp. 1792–1800.

177. GOJCIC, Zan; ZHOU, Caifa; WEGNER, Jan D; WIESER, Andreas. The perfect match: 3d point cloud matching with smoothed densities. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5545–5554.

178. TOMASI, Carlo; MANDUCHI, Roberto. Bilateral filtering for gray and color images. In: *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*. 1998, pp. 839–846.

179. LOGANAYAGI, T; KASHWAN, KR. A robust edge preserving bilateral filter for ultrasound kidney image. *Indian Journal of Science and Technology*. 2015, vol. 8, no. 23, p. 1.

180. YU, Haiping; HE, Fazhi; PAN, Yiteng. A scalable region-based level set method using adaptive bilateral filter for noisy image segmentation. *Multimedia Tools and Applications*. 2020, vol. 79, no. 9, pp. 5743–5765.

181. NADERNEJAD, Ehsan; SHARIFZADEH, Sara. A new method for image segmentation based on Fuzzy C-means algorithm on pixonal images formed by bilateral filtering. *Signal, Image and Video Processing*. 2013, vol. 7, no. 5, pp. 855–863.

182. ZOU, Bochang; QIU, Huadong; LU, Yufeng. Point cloud reduction and denoising based on optimized downsampling and bilateral filtering. *Ieee Access*. 2020, vol. 8, pp. 136316–136326.

183. ALMIRA, Gusti Ayu et al. Performance analysis of Gaussian and bilateral filter in case of determination the fetal length. In: *2016 International Conference on Knowledge Creation and Intelligent Computing (KCIC)*. 2016, pp. 246–252.

184. BANTERLE, Francesco; CORSINI, Massimiliano; CIGNONI, Paolo; SCOPIGNO, Roberto. A low-memory, straightforward and fast bilateral filter through subsampling in spatial domain. In: *Computer Graphics Forum*. 2012, vol. 31, pp. 19–32. No. 1.

185. DAI, Longquan; YUAN, Mengke; ZHANG, Xiaopeng. Speeding up the bilateral filter: A joint acceleration way. *IEEE Transactions on Image Processing*. 2016, vol. 25, no. 6, pp. 2657–2672.

186. JANG, Sung-Joon; HWANG, Youngbae. Noise-aware and light-weight VLSI design of bilateral filter for robust and fast image denoising in mobile systems. *Sensors*. 2020, vol. 20, no. 17, p. 4722.

187. GAVASKAR, Ruturaj G; CHAUDHURY, Kunal N. Fast adaptive bilateral filtering. *IEEE Transactions on Image Processing*. 2018, vol. 28, no. 2, pp. 779–790.

188. KULIKAJEVAS, Audrius; MASKELIŪNAS, Rytis; DAMAŠEVIČIUS, Robertas; WLODARCZYK-SIELICKA, Marta. Auto-refining reconstruction algorithm for recreation of limited angle humanoid depth data. *Sensors*. 2021, vol. 21, no. 11, p. 3702.

189. DICE, Lee R. Measures of the amount of ecologic association between species. *Ecology*. 1945, vol. 26, no. 3, pp. 297–302.

190. JACCARD, Paul. The distribution of the flora in the alpine zone. 1. *New phytologist*. 1912, vol. 11, no. 2, pp. 37–50.

191. EELBODE, Tom et al. Optimization for medical image segmentation: theory and practice when evaluating with dice score or Jaccard index. *IEEE Transactions on Medical Imaging*. 2020, vol. 39, no. 11, pp. 3679–3690.

192. CAPPABIANCO, Fábio AM; MIRANDA, Paulo AV de; UDUPA, Jayaram K. A critical analysis of the methods of evaluating MRI brain segmentation algorithms. In: *2017 IEEE international conference on image processing (ICIP)*. 2017, pp. 3894–3898.

193. BURT, Andrew; DISNEY, Mathias; CALDERS, Kim. Extracting individual trees from lidar point clouds using treeseg. *Methods in Ecology and Evolution*. 2019, vol. 10, no. 3, pp. 438–445.

194. LIU, Zongming; LIU, Guodong; LI, Jianxun; YE, Dong. Pose estimation of rigid object in point cloud. In: *2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. 2016, pp. 708–713.

195. NEUBAUER, Kai. *Benchmarking SLAM reconstructions in absence of a complete ground truth*. 2022. PhD thesis. University of British Columbia.

196. LIU, Ying; ZHONG, Ruofei. Buildings and terrain of urban area point cloud segmentation based on PCL. In: *IOP Conference Series: Earth and Environmental Science*. 2014, vol. 17, p. 012238. No. 1.

197. NGUYEN, Anh; CANO, Abraham Monrroy; EDAHIRO, Masato; KATO, Shinpei. Fast Euclidean Cluster Extraction Using GPUs. *Journal of Robotics and Mechatronics*. 2020, vol. 32, no. 3, pp. 548–560.

198. LIU, Xuan; HU, Chunhua; LI, Pingping. Automatic segmentation of overlapped poplar seedling leaves combining mask R-CNN and DBSCAN. *Computers and Electronics in Agriculture*. 2020, vol. 178, p. 105753.

199. SCHAUER, Johannes; NÜCHTER, Andreas. Collision detection between point clouds using an efficient kd tree implementation. *Advanced Engineering Informatics*. 2015, vol. 29, no. 3, pp. 440–458.

200. YANG, Bo et al. Learning object bounding boxes for 3D instance segmentation on point clouds. *Advances in neural information processing systems*. 2019, vol. 32.

201. ALI, Waleed et al. Yolo3d: End-to-end real-time 3d oriented object bounding box detection from lidar point cloud. In: *Proceedings of the European conference on computer vision (ECCV) workshops*. 2018, pp. 0–0.

202. XU, Danfei; ANGUELOV, Dragomir; JAIN, Ashesh. Pointfusion: Deep sensor fusion for 3d bounding box estimation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 244–253.

203. ZHOU, Yin; TUZEL, Oncel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4490–4499.

204. HE, Chenhang et al. Structure aware single-stage 3d object detection from point cloud. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11873–11882.

205. LI, Bo. 3d fully convolutional network for vehicle detection in point cloud. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 1513–1518.

206. SHI, Weijing; RAJKUMAR, Raj. Point-gnn: Graph neural network for 3d object detection in a point cloud. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 1711–1719.

207. GEIGER, Andreas; LENZ, Philip; STILLER, Christoph; URTASUN, Raquel. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*. 2013, vol. 32, no. 11, pp. 1231–1237.

208. WAQAS ZAMIR, Syed et al. isaid: A large-scale dataset for instance segmentation in aerial images. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019, pp. 28–37.

209. SAHA, Baidya Nath et al. Quick detection of brain tumors and edemas: A bounding box method using symmetry. *Computerized medical imaging and graphics*. 2012, vol. 36, no. 2, pp. 95–107.

210. NEBEHAY, Georg; PFLUGFELDER, Roman. Clustering of static-adaptive correspondences for deformable object tracking. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 2784–2791.

211. KEUPER, Margret et al. Motion segmentation & multiple object tracking by correlation co-clustering. *IEEE transactions on pattern analysis and machine intelligence*. 2018, vol. 42, no. 1, pp. 140–153.

212. WANG, Xinlong et al. Solov2: Dynamic and fast instance segmentation. *Advances in Neural information processing systems*. 2020, vol. 33, pp. 17721–17732.

213. WANG, Qiang et al. Fast online object tracking and segmentation: A unifying approach. In: *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*. 2019, pp. 1328–1338.

214. HURTIK, Petr et al. Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3. *Neural Computing and Applications*. 2022, vol. 34, no. 10, pp. 8275–8290.

215. SHAFII, Mahnaz; SID-AHMED, Maher. Skew detection and correction based on an axes-parallel bounding box. *International Journal on Document Analysis and Recognition (IJDAR)*. 2015, vol. 18, no. 1, pp. 59–71.

216. HSU, Cheng-Chun et al. Weakly supervised instance segmentation using the bounding box tightness prior. *Advances in Neural Information Processing Systems*. 2019, vol. 32.

217. XU, Yongchao et al. Gliding vertex on the horizontal bounding box for multi-oriented object detection. *IEEE transactions on pattern analysis and machine intelligence*. 2020, vol. 43, no. 4, pp. 1452–1459.

218. JAIN, Suyog Dutt; GRAUMAN, Kristen. Predicting sufficient annotation strength for interactive foreground segmentation. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1313–1320.

219. HE, Yihui et al. Bounding box regression with uncertainty for accurate object detection. In: *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*. 2019, pp. 2888–2897.

220. DUBEY, Shiv Ram; DIXIT, Pushkar; SINGH, Nishant; GUPTA, Jay Prakash. Infected fruit part detection using K-means clustering segmentation technique. 2013.

221. DHANACHANDRA, Nameirakpam; MANGLEM, Khumanthem; CHANU, Yambem Jina. Image segmentation using K-means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*. 2015, vol. 54, pp. 764–771.

222. MITTAL, Himanshu et al. A comprehensive survey of image segmentation: clustering methods, performance parameters, and benchmark datasets. *Multimedia Tools and Applications*. 2022, vol. 81, no. 24, pp. 35001–35026.

223. ZERMAS, Dimitris; IZZAT, Izzat; PAPANIKOLOPOULOS, Nikolaos. Fast segmentation of 3d point clouds: A paradigm on lidar data for autonomous vehicle applications. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 5067–5073.

224. WANG, Ke et al. A portable and automatic Xtion-based measurement system for pig body size. *Computers and electronics in agriculture*. 2018, vol. 148, pp. 291–298.

225. SERRA, Jean Paul Frédéric. Image Analysis and Mathematical Morphology. In: 1983.

226. SHARMA, Arpit Kumar et al. Enhanced watershed segmentation algorithm-based modified ResNet50 model for brain tumor detection. *BioMed Research International*. 2022, vol. 2022.

227. YANG, Juntao et al. An individual tree segmentation method based on watershed algorithm and three-dimensional spatial distribution analysis from airborne LiDAR point clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*. 2020, vol. 13, pp. 1055–1067.

228. LEWIS, Samuel H; DONG, Aijuan. Detection of breast tumor candidates using marker-controlled watershed segmentation and morphological analysis. In: *2012 IEEE Southwest symposium on image analysis and interpretation*. 2012, pp. 1–4.

229. AYREY, Elias et al. Layer stacking: A novel algorithm for individual forest tree segmentation from LiDAR point clouds. *Canadian Journal of Remote Sensing*. 2017, vol. 43, no. 1, pp. 16–27.

230. MONGUS, Domen; ŽALIK, Borut. An efficient approach to 3D single tree-crown delineation in LiDAR data. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2015, vol. 108, pp. 219–233.

231. HUANG, Hongyu; LI, Xu; CHEN, Chongcheng. Individual tree crown detection and delineation from very-high-resolution UAV images based on bias field and marker-controlled watershed segmentation algorithms. *IEEE Journal of selected topics in applied earth observations and remote sensing*. 2018, vol. 11, no. 7, pp. 2253–2262.

232. KAUR, Amanpreet; VERMA, Ashish; SSIET, Derabassi. The marker-based watershed segmentation-a review. *IJEIT*. 2013, vol. 3, no. 3, pp. 171–174.

233. SINGHAI, Pratik P; LADHAKE, Siddharth A. Brain tumor detection using marker based watershed segmentation from digital mr images. *International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN*. 2013, pp. 2278–3075.

234. KWON, Goo-Rak et al. Brain image segmentation using a combination of expectation-maximization algorithm and watershed transform. *International Journal of Imaging Systems and Technology*. 2016, vol. 26, no. 3, pp. 225–232.

235. BOLELLI, Federico; ALLEGRETTI, Stefano; BARALDI, Lorenzo; GRANA, Costantino. Spaghetti labeling: Directed acyclic graphs for block-based connected components labeling. *IEEE Transactions on Image Processing*. 2019, vol. 29, pp. 1999–2012.

236. MACQUEEN, J. Classification and analysis of multivariate observations. In: *5th Berkeley Symp. Math. Statist. Probability*. 1967, pp. 281–297.

237. BHOLOWALIA, Purnima; KUMAR, Arvind. EBK-means: A clustering technique based on elbow method and k-means in WSN. *International Journal of Computer Applications*. 2014, vol. 105, no. 9.

238. SAĞLAM, Ali; MAKINECI, Hasan Bilgehan; BAYKAN, Ömer Kaan; BAYKAN, Nurdan Akhan. Clustering-based plane refitting of non-planar patches for voxel-based 3D point cloud segmentation using k-means clustering. *Traitement du Signal*. 2020.

239. ZHOU, Jing et al. Automated segmentation of soybean plants from 3D point cloud using machine learning. *Computers and Electronics in Agriculture*. 2019, vol. 162, pp. 143–153.

240. CHEN, Siyuan; TRUONG-HONG, Linh; LAEFER, Debra; MANGINA, Eleni. Automated bridge deck evaluation through UAV derived point cloud. In: *CERI-ITRN2018*. 2018, pp. 735–740.

241. YIN, Xuanyu; SASAKI, Yoko; WANG, Weimin; SHIMIZU, Kentaro. 3D Object Detection Method Based on YOLO and K-Means for Image and Point Clouds. *arXiv preprint arXiv:2005.02132*. 2020.

242. GRIMM, Oliver et al. Amygdalar and hippocampal volume: a comparison between manual segmentation, Freesurfer and VBM. *Journal of neuroscience methods*. 2015, vol. 253, pp. 254–261.

243. SCHOEMAKER, Dorothee et al. Hippocampus and amygdala volumes from magnetic resonance images in children: Assessing accuracy of FreeSurfer and FSL against manual segmentation. *Neuroimage*. 2016, vol. 129, pp. 1–14.

244. NUGENT, Allison C et al. Automated subcortical segmentation using FIRST: test–retest reliability, interscanner reliability, and comparison to manual segmentation. *Human brain mapping*. 2013, vol. 34, no. 9, pp. 2313–2329.

245. SALVAGGIO, Giuseppe et al. Deep learning network for segmentation of the prostate gland with median lobe enlargement in T2-weighted mr images: comparison with manual segmentation method. *Current Problems in Diagnostic Radiology*. 2021.

246. MAGLIARO, Chiara; CALLARA, Alejandro L; VANELLO, Nicola; AHLUWALIA, Arti. A manual segmentation tool for three-dimensional neuron datasets. *Frontiers in neuroinformatics*. 2017, vol. 11, p. 36.

247. WILD, Daniel; WEBER, Maximilian; EGGER, Jan. Client/server based online environment for manual segmentation of medical images. *arXiv preprint arXiv:1904.08610*. 2019.

248. CHARTRAND, Gabriel et al. Semi-automated liver CT segmentation using Laplacian meshes. In: *2014 IEEE 11th International Symposium on Biomedical Imaging (ISBI)*. 2014, pp. 641–644.

249. PIRI, Reza et al. Aortic wall segmentation in 18F-sodium fluoride PET/CT scans: Head-to-head comparison of artificial intelligence-based versus manual segmentation. *Journal of Nuclear Cardiology*. 2021, pp. 1–10.

250. CARDENAS, Carlos E et al. Advances in auto-segmentation. In: *Seminars in radiation oncology*. 2019, vol. 29, pp. 185–197. No. 3.

251. MCGRATH, Hari et al. Manual segmentation versus semi-automated segmentation for quantifying vestibular schwannoma volume on MRI. *International journal of computer assisted radiology and surgery*. 2020, vol. 15, no. 9, pp. 1445–1455.

252. BECKER, Anton S et al. Variability of manual segmentation of the prostate in axial T2-weighted MRI: A multi-reader study. *European journal of radiology*. 2019, vol. 121, p. 108716.

253. YAMASHITA, Rikiya et al. Radiomic feature reproducibility in contrast-enhanced CT of the pancreas is affected by variabilities in scan parameters and manual segmentation. *European radiology*. 2020, vol. 30, no. 1, pp. 195–205.

254. SULTANA, Farhana; SUFIAN, Abu; DUTTA, Paramartha. Evolution of image segmentation using deep convolutional neural network: A survey. *Knowledge-Based Systems*. 2020, vol. 201, p. 106062.

255. GIRSHICK, Ross; DONAHUE, Jeff; DARRELL, Trevor; MALIK, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

256. UIJLINGS, Jasper RR; VAN DE SANDE, Koen EA; GEVERS, Theo; SMEULDERS, Arnold WM. Selective search for object recognition. *International journal of computer vision*. 2013, vol. 104, no. 2, pp. 154–171.

257. ZENG, Liang et al. A novel region-based image registration method for multisource remote sensing images via CNN. *IEEE journal of selected topics in applied earth observations and remote sensing*. 2020, vol. 14, pp. 1821–1831.

258. GIRSHICK, Ross. Fast r-cnn. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

259. HE, Kaiming; GKIOXARI, Georgia; DOLLÁR, Piotr; GIRSHICK, Ross. Mask r-cnn. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

260. KAMNITSAS, Konstantinos et al. Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. *Medical image analysis*. 2017, vol. 36, pp. 61–78.

261. BASHA, SH Shabbeer; DUBEY, Shiv Ram; PULABAIGARI, Viswanath; MUKHERJEE, Snehasis. Impact of fully connected layers on performance of convolutional neural networks for image classification. *Neurocomputing*. 2020, vol. 378, pp. 112–119.

262. DAI, Jifeng; LI, Yi; HE, Kaiming; SUN, Jian. R-fcn: Object detection via region-based fully convolutional networks. *Advances in neural information processing systems*. 2016, vol. 29.

263. BRIDLE, John. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Advances in neural information processing systems*. 1989, vol. 2.

264. FUKUSHIMA, Kunihiko. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*. 1975, vol. 20, no. 3, pp. 121–136.

265. AGARAP, Abien Fred. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*. 2018.

266. JI, Yuzhu; ZHANG, Haijun; ZHANG, Zhao; LIU, Ming. CNN-based encoder-decoder networks for salient object detection: A comprehensive review and recent advances. *Information Sciences*. 2021, vol. 546, pp. 835–857.

267. RONNEBERGER, O.; FISCHER, P.; BROX, T. *U-net: Convolutional networks for biomedical image segmentation*. 2015. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

268. MAZAHERI, Ghazal; MITHUN, Niluthpol Chowdhury; BAPPY, Jawadul H; ROY-CHOWDHURY, Amit K. A Skip Connection Architecture for Localization of Image Manipulations. In: *CVPR workshops*. 2019, pp. 119–129.

269. YOU, Chenyu; YANG, Linfeng; ZHANG, Yi; WANG, Ge. Low-dose CT via deep CNN with skip connection and network-in-network. In: *Developments in X-Ray tomography XII*. 2019, vol. 11113, pp. 429–434.

270. PENG, Yali et al. Dilated residual networks with symmetric skip connection for image denoising. *Neurocomputing*. 2019, vol. 345, pp. 67–76.

271. FENG, Ruicheng et al. Removing diffraction image artifacts in under-display camera via dynamic skip connection network. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 662–671.

272. WANG, Haonan; CAO, Peng; WANG, Jiaqi; ZAIANE, Osmar R. Uctransnet: rethinking the skip connections in u-net from a channel-wise perspective with transformer. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2022, vol. 36, pp. 2441–2449. No. 3.

273. CAMALAN, Seda et al. Gender detection using 3d anthropometric measurements by kinect. *Metrology and Measurement Systems*. 2018, vol. 25, no. 2.

274. ZHAO, Zhong-Qiu; ZHENG, Peng; XU, Shou-tao; WU, Xindong. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*. 2019, vol. 30, no. 11, pp. 3212–3232.

275. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*. 2012, vol. 25.

276. ALOM, Md Zahangir et al. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*. 2018.

277. LECUN, Yann et al. Backpropagation applied to handwritten zip code recognition. *Neural computation*. 1989, vol. 1, no. 4, pp. 541–551.

278. ISLAM, Md Rakibul; MATIN, Abdul. Detection of COVID 19 from CT image by the novel LeNet-5 CNN architecture. In: *2020 23rd International Conference on Computer and Information Technology (ICCIT)*. 2020, pp. 1–5.

279. SITAULA, Chiranjibi; HOSSAIN, Mohammad Belayet. Attention-based VGG-16 model for COVID-19 chest X-ray image classification. *Applied Intelligence*. 2021, vol. 51, no. 5, pp. 2850–2863.

280. GUAN, Qing et al. Deep convolutional neural network VGG-16 model for differential diagnosing of papillary thyroid carcinomas in cytological images: a pilot study. *Journal of Cancer*. 2019, vol. 10, no. 20, p. 4876.

281. GENG, Lei; ZHANG, Siqi; TONG, Jun; XIAO, Zhitao. Lung segmentation method with dilated convolution based on VGG-16 network. *Computer Assisted Surgery*. 2019, vol. 24, no. sup2, pp. 27–33.

282. YU, Wei et al. Visualizing and comparing AlexNet and VGG using deconvolutional layers. In: *33 rd International Conference on Machine Learning*. 2016.

283. CANZIANI, Alfredo; PASZKE, Adam; CULURCIELLO, Eugenio. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*. 2016.

284. ALQAZZAZ, Salma; SUN, Xianfang; YANG, Xin; NOKES, Len. Automated brain tumor segmentation on multi-modal MR image using SegNet. *Computational Visual Media*. 2019, vol. 5, no. 2, pp. 209–219.

285. CHEN, Tingyang et al. Pavement crack detection and recognition using the architecture of segNet. *Journal of Industrial Information Integration*. 2020, vol. 18, p. 100144.

286. ALONSO, Inigo; MURILLO, Ana C. EV-SegNet: Semantic segmentation for event-based cameras. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019, pp. 0–0.

287. ZHU, Fengcheng et al. Automatic measurement of fetal femur length in ultrasound images: a comparison of random forest regression model and SegNet. *Mathematical Biosciences and Engineering*. 2021, vol. 18, no. 6, pp. 7790–7805.

288. MOU, Lichao; HUA, Yuansheng; ZHU, Xiao Xiang. A relation-augmented fully convolutional network for semantic segmentation in aerial scenes. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12416–12425.

289. SEO, Hyunseok et al. Modified U-Net (mU-Net) with incorporation of object-dependent high level features for improved liver and liver-tumor segmentation in CT images. *IEEE transactions on medical imaging*. 2019, vol. 39, no. 5, pp. 1316–1325.

290. IGLOVIKOV, Vladimir; SHVETS, Alexey. Ternausnet: U-net with vgg11 encoder pre-trained on imagenet for image segmentation. *arXiv preprint arXiv:1801.05746*. 2018.

291. CHEN, Liang-Chieh et al. Encoder-decoder with atrous separable convolution for semantic image segmentation. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 801–818.

292. ZHOU, Sihang et al. High-resolution encoder–decoder networks for low-contrast medical image segmentation. *IEEE Transactions on Image Processing*. 2019, vol. 29, pp. 461–475.

293. ZHOU, Zongwei; RAHMAN SIDDIQUEE, Md Mahfuzur; TAJBAKHSH, Nima; LIANG, Jianming. Unet++: A nested u-net architecture for medical image segmentation. In: *Deep learning in medical image analysis and multimodal learning for clinical decision support*. Springer, 2018, pp. 3–11.

294. SUN, Yi; TIAN, Yan; XU, Yiping. Problems of encoder-decoder frameworks for high-resolution remote sensing image segmentation: Structural stereotype and insufficient learning. *Neurocomputing*. 2019, vol. 330, pp. 297–304.

295. AICH, Shubhra; KAMP, William van der; STAVNESS, Ian. Semantic binary segmentation using convolutional networks without decoders. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 197–201.

296. QIAO, Maoying; CHENG, Jun; BIAN, Wei; TAO, Dacheng. Biview learning for human posture segmentation from 3D points cloud. *PloS one*. 2014, vol. 9, no. 1, e85811.

297. SHEN, Yi et al. Empirical comparisons of deep learning networks on liver segmentation. *Computers, Materials and Continua*. 2020, vol. 62, no. 3, pp. 1233–1247.

298. IGLOVIKOV, Vladimir; SEFERBEKOV, Selim; BUSLAEV, Alexander; SHVETS, Alexey. Ternausnetv2: Fully convolutional network for instance segmentation. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 233–237.

299. PENNISI, Andrea et al. Skin lesion image segmentation using Delaunay Triangulation for melanoma detection. *Computerized Medical Imaging and Graphics*. 2016, vol. 52, pp. 89–103.

300. WANG, Guotai; LI, Wenqi; OURSELIN, Sébastien; VERCAUTEREN, Tom. Automatic brain tumor segmentation using cascaded anisotropic convolutional neural networks. In: *International MICCAI brainlesion workshop*. 2017, pp. 178–190.

301. SONG, Kyungmin; CHOI, Han-Soo; KANG, Myungjoo. Squeezed fire binary segmentation model using convolutional neural network for outdoor images on embedded device. *Machine Vision and Applications*. 2021, vol. 32, no. 6, pp. 1–12.

302. DENG, Tengfang et al. Comparison of multi-class and fusion of multiple single-class SegNet model for mapping karst wetland vegetation using UAV images. *Scientific Reports*. 2022, vol. 12, no. 1, p. 13270.

303. GROS, Charley; LEMAY, Andreanne; COHEN-ADAD, Julien. SoftSeg: Advantages of soft versus binary training for image segmentation. *Medical image analysis*. 2021, vol. 71, p. 102038.

304. HE, K.; ZHANG, X.; REN, S.; SUN, J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2015, vol. 37, no. 9, pp. 1904–1916.

305. DANGI, Shusil; LINTE, Cristian A; YANIV, Ziv. A distance map regularized CNN for cardiac cine MR image segmentation. *Medical physics*. 2019, vol. 46, no. 12, pp. 5637–5651.

306. VISIN, F. et al. ReSeg: A Recurrent Neural Network-Based Model for Semantic Segmentation. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 2016, pp. 426–433.

307. SHUAI, B.; ZUO, Z.; WANG, B.; WANG, G. DAG-Recurrent Neural Networks for Scene Labeling. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2016, vol. 2016-December, pp. 3620–3629.

308. SONG, Liangchen; YU, Gang; YUAN, Junsong; LIU, Zicheng. Human pose estimation and its application to action recognition: A survey. *Journal of Visual Communication and Image Representation*. 2021, vol. 76, p. 103055.

309. YAO, Rui et al. Video object segmentation and tracking: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*. 2020, vol. 11, no. 4, pp. 1–47.

310. LONG, Jonathan; SHELHAMER, Evan; DARRELL, Trevor. Fully convolutional networks for semantic segmentation. In: *IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

311. BREIMAN, Leo. Random forests. *Machine learning*. 2001, vol. 45, no. 1, pp. 5–32.

312. CRIMINISI, Antonio; SHOTTON, Jamie. *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media, 2013.

313. CUTLER, Adele; CUTLER, D Richard; STEVENS, John R. Random forests. In: *Ensemble machine learning*. Springer, 2012, pp. 157–175.

314. WANG, Sutong et al. An improved random forest-based rule extraction method for breast cancer diagnosis. *Applied Soft Computing*. 2020, vol. 86, p. 105941.

315. FAWAGREH, Khaled; GABER, Mohamed Medhat; ELYAN, Eyad. Random forests: from early developments to recent advancements. *Systems Science & Control Engineering: An Open Access Journal*. 2014, vol. 2, no. 1, pp. 602–609.

316. MAHAPATRA, Dwarikanath. Analyzing training information from random forests for improved image segmentation. *IEEE Transactions on Image Processing*. 2014, vol. 23, no. 4, pp. 1504–1512.

317. MAHAPATRA, Dwarikanath. Automatic cardiac segmentation using semantic information from random forests. *Journal of digital imaging*. 2014, vol. 27, no. 6, pp. 794–804.

318. MAHAPATRA, Dwarikanath. Graph cut based automatic prostate segmentation using learned semantic information. In: *2013 IEEE 10th International Symposium on Biomedical Imaging*. 2013, pp. 1316–1319.

319. WEI, Jing et al. Cloud detection for Landsat imagery by combining the random forest and superpixels extracted via energy-driven sampling segmentation approaches. *Remote Sensing of Environment*. 2020, vol. 248, p. 112005.

320. HERNÁNDEZ-VELA, Antonio et al. Graph cuts optimization for multi-limb human segmentation in depth maps. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 726–732.

321. CUINGNET, Rémi et al. Automatic detection and segmentation of kidneys in 3D CT images using random forests. In: *International conference on medical image computing and computer-assisted intervention*. 2012, pp. 66–74.

322. NI, Huan; LIN, Xiangguo; ZHANG, Jixian. Classification of ALS point cloud with improved point cloud segmentation and random forests. *Remote Sensing*. 2017, vol. 9, no. 3, p. 288.

323. YANG, Tiejun; SONG, Jikun; LI, Lei. A deep learning model integrating SK-TPCNN and random forests for brain tumor segmentation in MRI. *Biocybernetics and Biomedical Engineering*. 2019, vol. 39, no. 3, pp. 613–623.

324. PEREIRA, Sergio et al. Automatic brain tissue segmentation in MR images using random forests and conditional random fields. *Journal of neuroscience methods*. 2016, vol. 270, pp. 111–123.

325. WU, Yaokun; MISRA, Siddharth. Intelligent image segmentation for organic-rich shales using random forest, wavelet transform, and hessian matrix. *IEEE Geoscience and Remote Sensing Letters*. 2019, vol. 17, no. 7, pp. 1144–1147.

326. SOLTANINEJAD, Mohammadreza et al. MRI brain tumor segmentation and patient survival prediction using random forests and fully convolutional networks. In: *International MICCAI brainlesion workshop*. 2017, pp. 204–215.

327. LEE, Der-Tsai; WONG, Chak-Kuen. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*. 1977, vol. 9, no. 1, pp. 23–29.

328. JIANG, Yuming et al. JointRCNN: a region-based convolutional neural network for optic disc and cup segmentation. *IEEE Transactions on Biomedical Engineering*. 2019, vol. 67, no. 2, pp. 335–343.

329. ZHU, Yukun; URTASUN, Raquel; SALAKHUTDINOV, Ruslan; FIDLER, Sanja. segdeepm: Exploiting segmentation and context in deep neural networks for object detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 4703–4711.

330. SULTANA, Farhana; SUFIAN, Abu; DUTTA, Paramartha. A review of object detection models based on convolutional neural network. *Intelligent Computing: Image Processing Based Applications*. 2020, pp. 1–16.

331. GASHLER, Mike. *What is the equivalent nth_element function in Java?* [Online] [visited on 2022-06-26]. Available from: `https://stackoverflow.com/a/47702047/2273107`.

332. SCHIMPL, Michaela et al. Association between walking speed and age in healthy, free-living individuals using mobile accelerometry—a cross-sectional study. *PloS one*. 2011, vol. 6, no. 8, e23299.

333. YU, Yang et al. Performance analysis and optimization of full garbage collection in memory-hungry environments. *ACM SIGPLAN Notices*. 2016, vol. 51, no. 7, pp. 123–130.

334. SAUNDERS, Richard. Sunyaev-Zel'dovich observations with the Ryle Telescope. *Microwave background anisotropies*. 1997, pp. 377–381.

335. ADAMS, Douglas. *The Restaurant At The End Of The Universe*. London: Pan Books, 1980.

336. AGARWAL, Mohit; GUPTA, Suneet; BISWAS, KK. A new Conv2D model with modified ReLU activation function for identification of disease type and severity in cucumber plant. *Sustainable Computing: Informatics and Systems*. 2021, vol. 30, p. 100473.

337. SCHMIDT, Robin M; SCHNEIDER, Frank; HENNIG, Philipp. Descending through a crowded valley-benchmarking deep learning optimizers. In: *International Conference on Machine Learning*. 2021, pp. 9367–9376.

338. YI, Dokkyun; AHN, Jaehyun; JI, Sangmin. An effective optimization method for machine learning based on ADAM. *Applied Sciences*. 2020, vol. 10, no. 3, p. 1073.

339. CHEN, Xiangyi; LIU, Sijia; SUN, Ruoyu; HONG, Mingyi. On the convergence of a class of adam-type algorithms for non-convex optimization. *arXiv preprint arXiv:1808.02941*. 2018.

340. RUBY, Usha; YENDAPALLI, Vamsidhar. Binary cross entropy with deep learning technique for image classification. *Int. J. Adv. Trends Comput. Sci. Eng*. 2020, vol. 9, no. 10.

341. CRESWELL, Antonia; ARULKUMARAN, Kai; BHARATH, Anil A. On denoising autoencoders trained to minimise binary cross-entropy. *arXiv preprint arXiv:1708.08487*. 2017.

342. *binary-trees - Which programs are fastest? (Benchmarks Game)* [online] [visited on 2023-03-10]. Available from: `https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/binarytrees.html`.

343. NOWAK, Tomasz. *GitHub - toimcio/SegNet-tensorflow* [online]. 2018 [visited on 2022-10-30]. Available from: `https://github.com/toimcio/SegNet-tensorflow`.

344. KORNILOV, Anton S; SAFONOV, Ilia V. An overview of watershed algorithm implementations in open source libraries. *Journal of Imaging*. 2018, vol. 4, no. 10, p. 123.

345. KARASTERGIOU, Kalypso; SMITH, Steven R; GREENBERG, Andrew S; FRIED, Susan K. Sex differences in human adipose tissues–the biology of pear shape. *Biology of sex differences*. 2012, vol. 3, no. 1, pp. 1–12.

346. *SegNet-tensorflow* [online]. 2018 [visited on 2022-12-18]. Available from: `https://github.com/toimcio/SegNet-tensorflow`.

Curriculum Vitae

PERSONAL INFORMATION

Karolis Ryselis

Draugystės str. 3C-40, Kaunas, LT-51281, Lithuania

+370 630 88 154

Karolis.ryselis@ktu.edu

Sex Male | Date of birth 06/Oct/1992 | Nationality Lithuanian

WORK EXPERIENCE

July 2012 - present

Software developer

Esperonus

- Creating software architecture
- Backend programming
- Python / Django, Javascript

Business / sector Information Technology

October 2014 – September 2018

Engineer

Kaunas University of Technology

- Developing small projects
- Android / C#

Business / sector Information Technology

September 2017 – present

Associate professor practitioner

Kaunas University of Technology

- Lectures of concurrent programming and software analysis & engineering tools
- Supervisor and committee member of Bachelor final projects

Business / sector Information Technology

EDUCATION AND TRAINING

2017-present

PhD in Informatics

Kaunas University of Technology

- Research area – framework for human body segmentation and motion analysis

2015-2017

Master in Software Engineering

Kaunas University of Technology

- Requirement analysis, architecture, testing
- Finished with a maximum possible grade of 10

2011-2015

Bachelor in Software Systems

Kaunas University of Technology

- Software development technologies, algorithms, software design patterns
- Finished with near perfect grade of 9.95

Mother tongue Lithuanian

Other languages

	UNDERSTANDING		SPEAKING		WRITING
	Listening	Reading	Spoken interaction	Spoken production	
English	C2	C2	C2	C2	C2
Russian	B1	B1	A2	A2	A2

Personal qualities

- Maximalist
- Perfectionist
- Persistent
- Love researching
- Able to work both alone and in team
- Good sense of humour

ADDITIONAL INFORMATION

Scientific publications

- Human position tracking algorithm using multiple Kinect devices and its research, IXI inter-university master and doctoral student conference "Information Society and University Studies", Lithuania, 2014.
- Non-standard human pose recognition precision using Kinect 2.0 sensors research, XX inter-university master and doctoral student conference "Information Society and University Studies", Lithuania, 2015.
- Efficient Hausdorff distance metric calculation algorithm for human position comparison to template and its performance and precision research, Proceedings of the IVUS International Conference on Information Technology, ceur-ws.org, 2017.
- Multiple Kinect based system to monitor and analyze key performance indicators of physical training, Hum. Cent. Comput. Inf. Sci., 2020.
- Computer-Aided Depth Video Stream Masking Framework for Human Body Segmentation in Depth Sensor Images, Sensors 22, 2022

Conferences

- Brain computing device application in a video game quality and usability research, IVUS International Conference on Information Technology, Lithuania, 2017.
- Efficient Human Motion Matching Algorithm for Depth Scanning Systems Based on Hausdorff Distance Metric, Data Analysis Methods for Software Systems, Druskininkai, Lithuania, 2018.

Exhibitions

- A.R. Drone and Kinect interface project, KTU Technorama 2012. (2012)
- Controlling devices using various platforms. Tank,, KTU Technorama 2013. (2013)
- Project "Vir²realism", KTU Technorama 2014 (2014)
- Virtual yoga trainer "Yogamin", KTU Technorama 2015 (2015) and GameOn (2015)
- Human pose comparison plugin for motion tracking systemsi, KTU Technorama 2016 (2016)
- Data merging system for motion tracking sensors, KTU Technorama 2017 (2017)

Other projects

- Rovio robot control using smart devices, (2012)
- Technical feasiblilty study on developing an application to activate EU-ALERT services in smartphones (2014)
- Kiosk software for Android devices "Kiosk-Mode" (2014-2015)
- Electricity and water readings recognition and declaration algorithmic prototype (2015-2016)

7. LIST OF PUBLICATIONS AND CONFERENCES

The results of this dissertation have been presented in 3 conferences (one of them, however, does not meet the criteria for PhD defense), and published in 3 journals indexed in *Web of Science*.

The conferences where the results were presented:

1. Ryselis K., Efficient Human Motion Matching Algorithm for Depth Scanning Systems Based on Hausdorff Distance Metric, 9th International Workshop on Data Analysis Methods for Software Systems, Druskininkai, 2017.
2. Ryselis K., Optimizations of searching three-dimensional trees for clustering (poster), FRUCT 31, Helsinki, 2022.
3. Ryselis, K. (2022). Random Forest Classifier for Correcting Point Cloud Segmentation Based on Metrics of Recursive 2-Means Splits. In: Lopata, A., Gudonienė, D., Butkienė, R. (eds.) Information and Software Technologies. ICIST 2022. Communications in Computer and Information Science, vol. 1665. Springer, Cham. https://doi.org/10.1007/978-3-031-16302-9_7

A poster was presented at the international FRUCT 31 conference. It outlined the performance improvements for the bounding box segmentation approach (Sections 3.4.1.5 and 4.2.4.1 of this dissertation). An article about random forest classifier based on the metrics of recursive 2-means splits was presented at the international ICIST 2022 conference (Sections 3.4.3 and 4.3 of this dissertation).

The results have also been published in the following *Web of Science* journals:

1. Ryselis, K., Petkus, T., Blažauskas, T. et al. Multiple Kinect based system to monitor and analyze key performance indicators of physical training. Hum. Cent. Comput. Inf. Sci. 10, 51 (2020). https://doi.org/10.1186/s13673-020-00256-4
2. Ryselis, K.; Blažauskas, T.; Damaševičius, R.; Maskeliūnas, R. Computer-Aided Depth Video Stream Masking Framework for Human Body Segmentation in Depth Sensor Images. Sensors 2022, 22, 3531. https://doi.org/10.3390/s22093531
3. Ryselis, K.; Blažauskas, T.; Damaševičius, R.; Maskeliūnas, R. Agrast-6: Abridged VGG-Based Reflected Lightweight Architecture for Binary Segmentation of Depth Images Captured by Kinect. Sensors 2022, 22, 6354. https://doi.org/10.3390/s22176354

All the articles are either in Q1 or Q2 (Sensors journal had impact factors of 3.576 and 3.847, Human-centric Computing and Information Sciences was showing 5.9000 at the time of publishing). “Multiple Kinect based system to monitor and analyze key performance indicators of physical training” presents an analysis of human joint load during training. The contribution from this dissertation if the framework to collect data, which is based on skeleton fusion methods presented in Section 3.7. “Computer-Aided Depth Video Stream Masking Framework for Human Body Segmentation in Depth Sensor Images” presents the whole software solution and methods to create it to reduce semi-automatic segmentation times, presented in Sections 3.4 and 4.2. Finally, “Agrast-6: Abridged VGG-Based Reflected Lightweight Architecture for Binary Segmentation of Depth Images Captured by Kinect” article presents the neural network described in Sections 3.5 and 4.4.

8. ACKNOWLEDGMENTS

The author would like to thank his scientific supervisor Prof. Tomas Blažauskas who provided ideas and has supervised for the past 11 years as well as Prof. Rytis Maskeliūnas for the scientific support.

The author would also like to thank his wife Justina, who helped with segmentation and took over with other duties while the author was busy with this dissertation. Another thanks goes to author's mother and the family of his wife for the moral support.

A. PROOF THAT TREES GROW FROM THE TOP

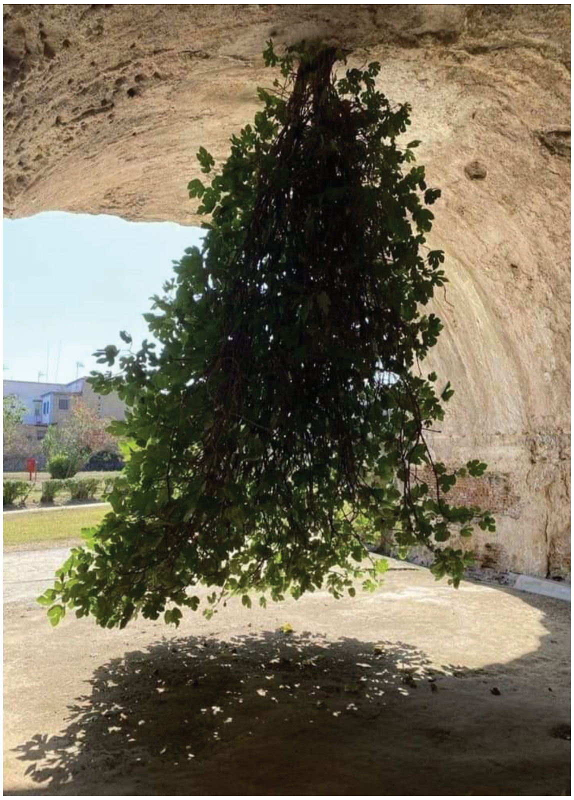


Figure A.1. An example of a real-life tree growing from the top[1]

[1]`https://www.reddit.com/r/ProgrammerHumor/comments/kk5ng1/finally_after_years_of_search_i_found_a_real_tree/`

B. ACTIVATIONS OF AGRAST-6 NEURAL NETWORK

This section shows activations of Agrast-6 neural network. Images with the least activations are skipped. The amounts of activations shown:

1. Layer 0: 1 / 1
2. Layer 1: 22 / 32
3. Layer 2: 27 / 32
4. Layer 3: 22 / 128
5. Layer 4: 43 / 128
6. Layer 6: 156 / 256
7. Layer 8: 240 / 256
8. Layer 9: 240 / 256
9. Layer 10: 108 / 128
10. Layer 11: 108 / 128
11. Layer 13: 1 / 1

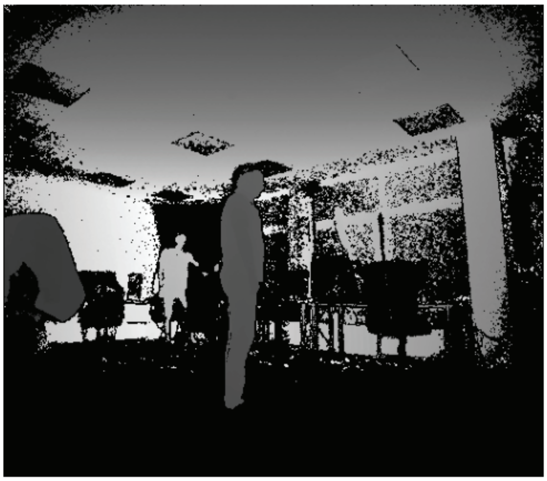


Figure B.1. Layer 0 (input) activation (input image)

Figure B.2. Layer 1 activations

Figure B.3. Layer 2 activations

Figure B.4. Layer 4 activations (part 1)

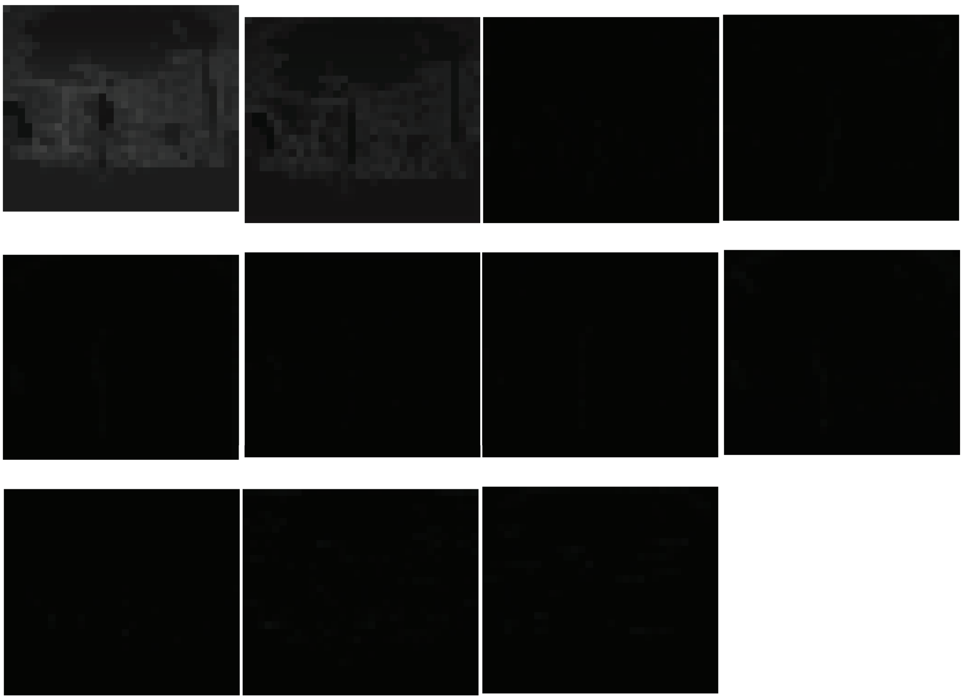


Figure B.5. Layer 4 activations (part 2)

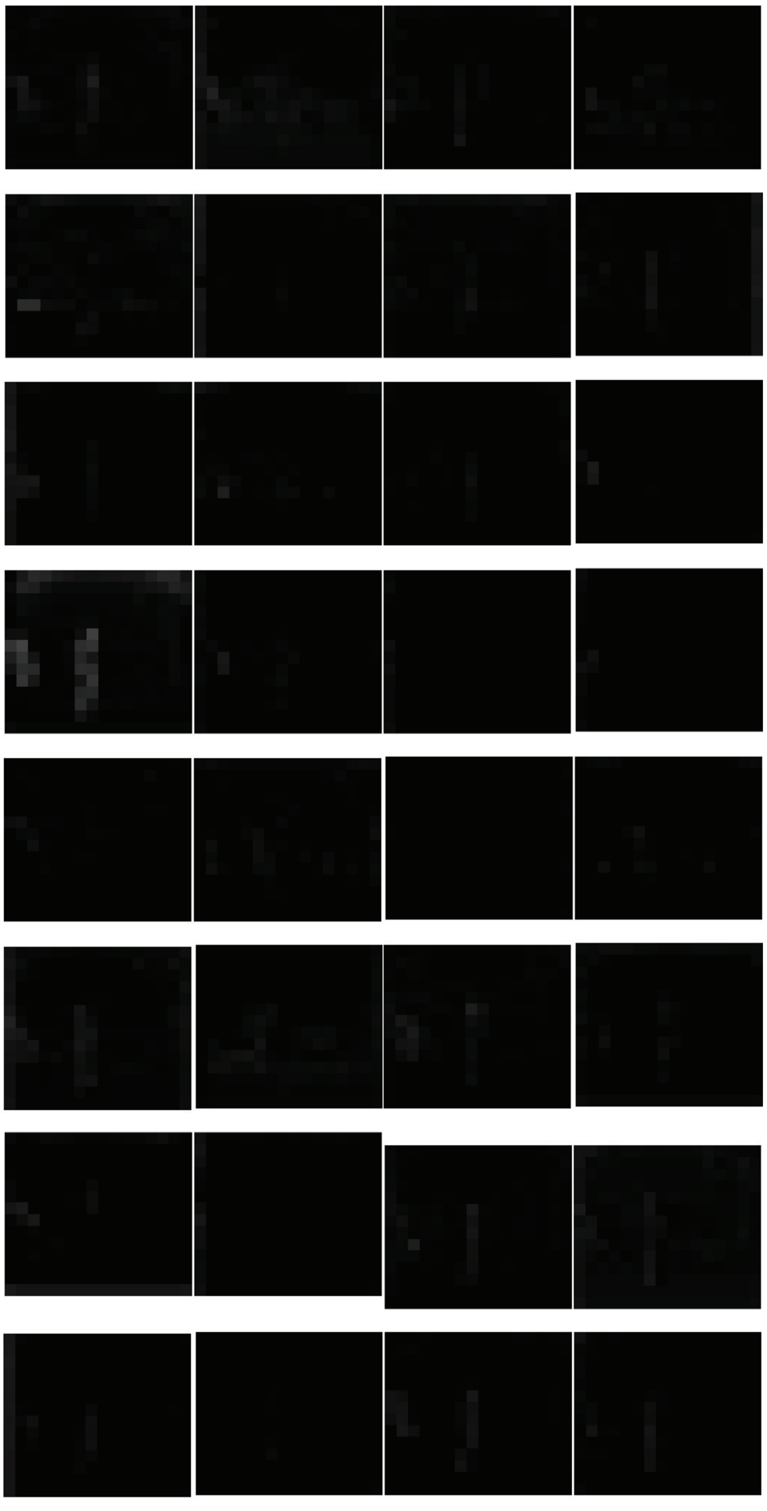


Figure B.6. Layer 6 activations (part 1)

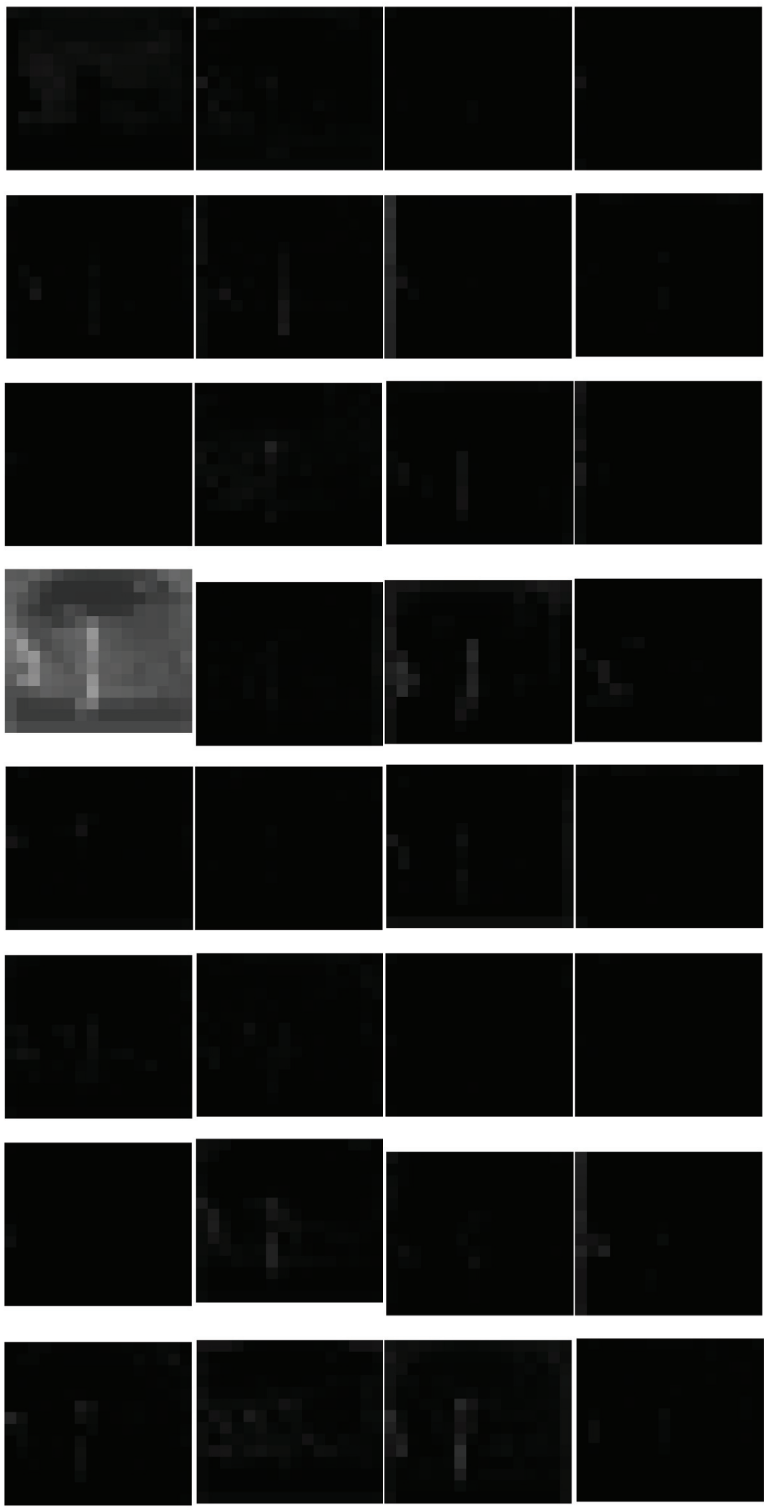


Figure B.7. Layer 6 activations (part 2)

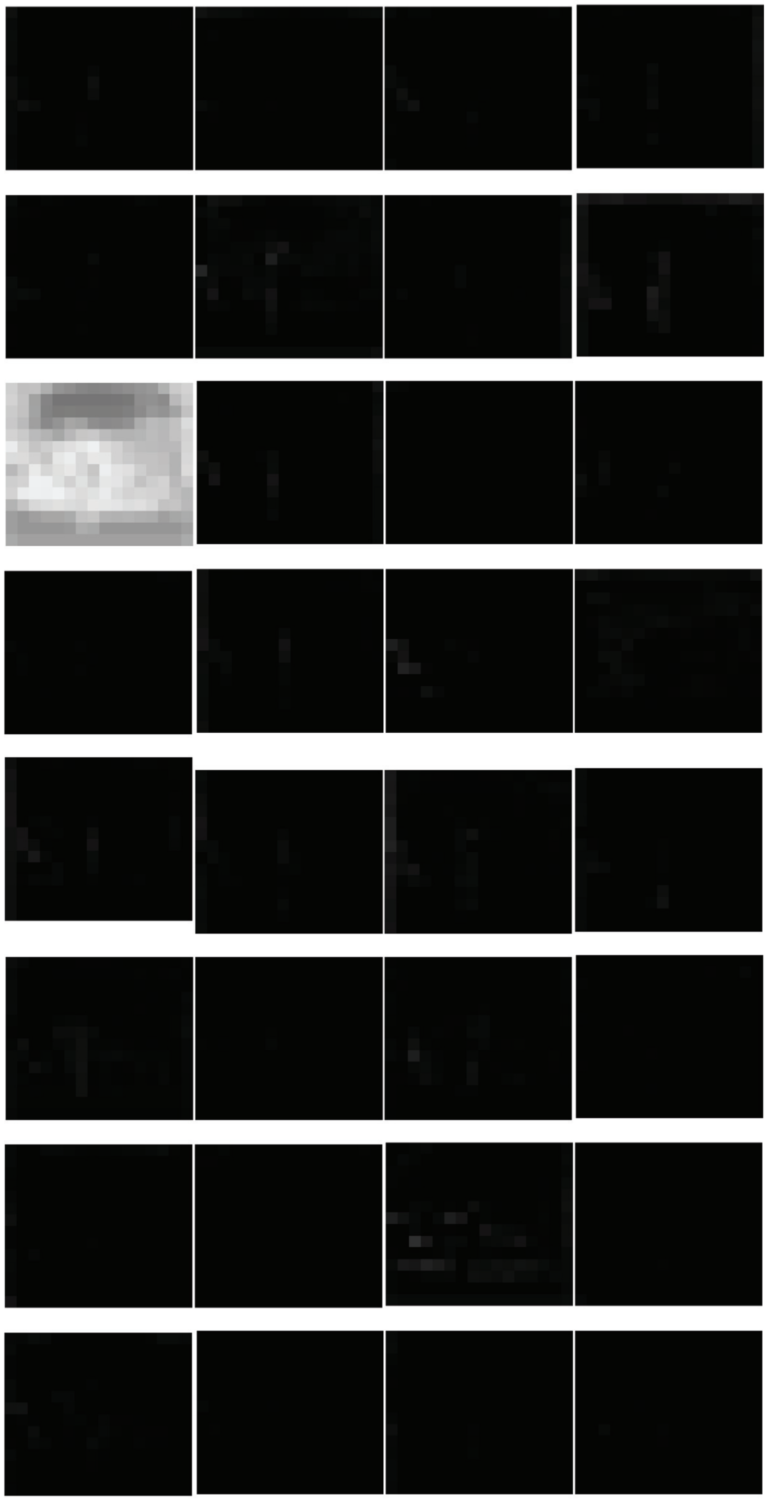


Figure B.8. Layer 6 activations (part 3)

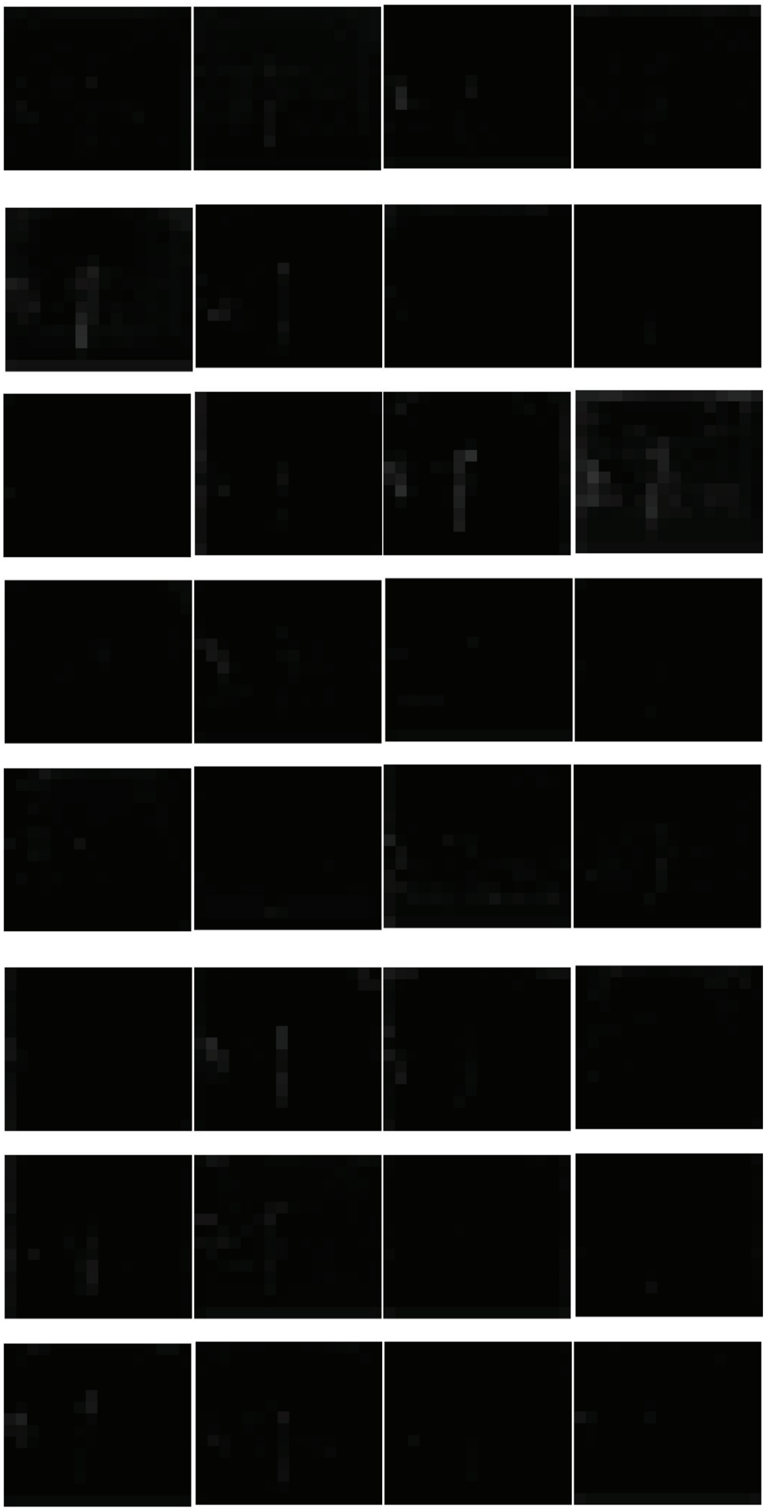


Figure B.9. Layer 6 activations (part 4)

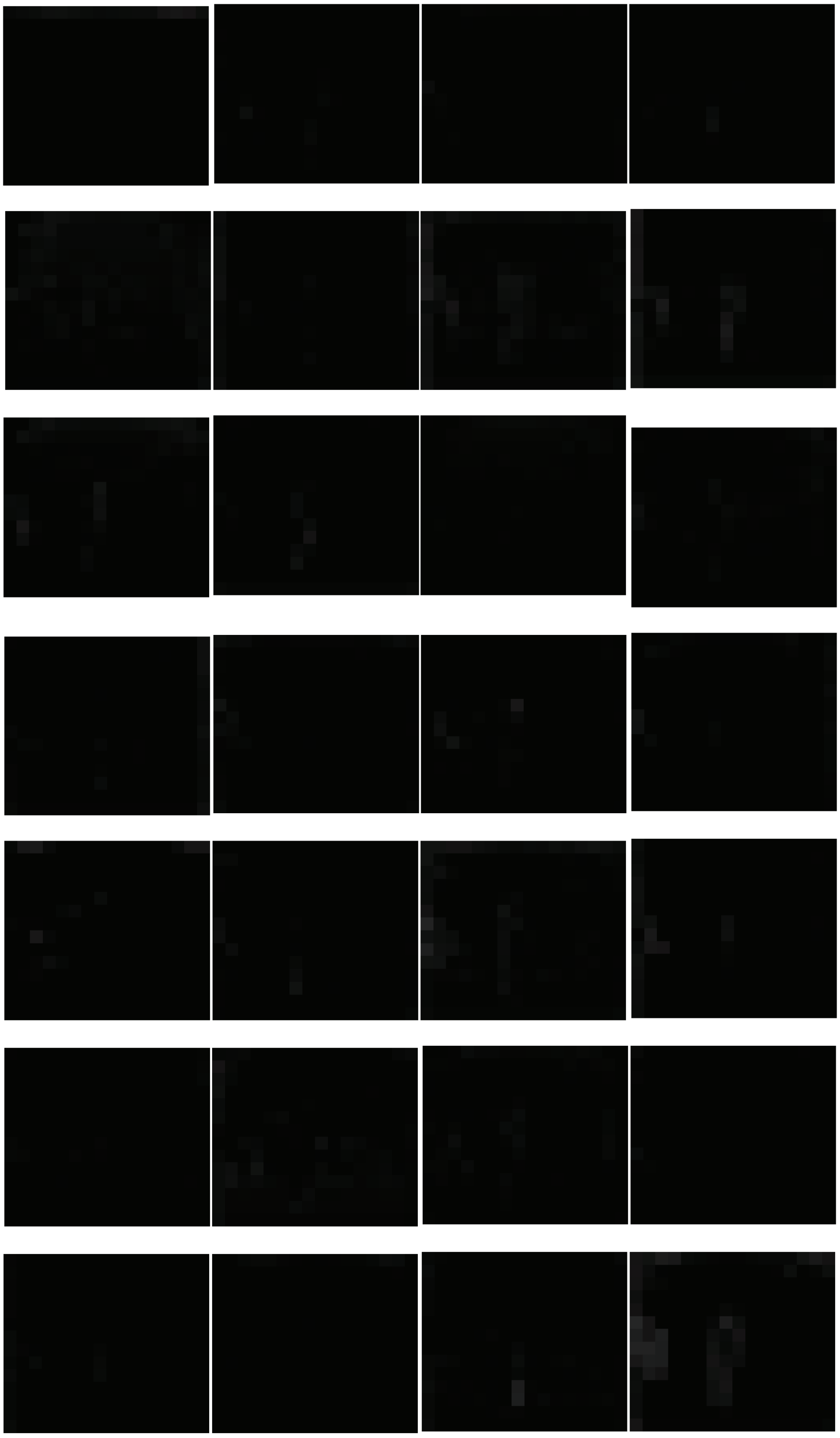


Figure B.10. Layer 6 activations (part 5)

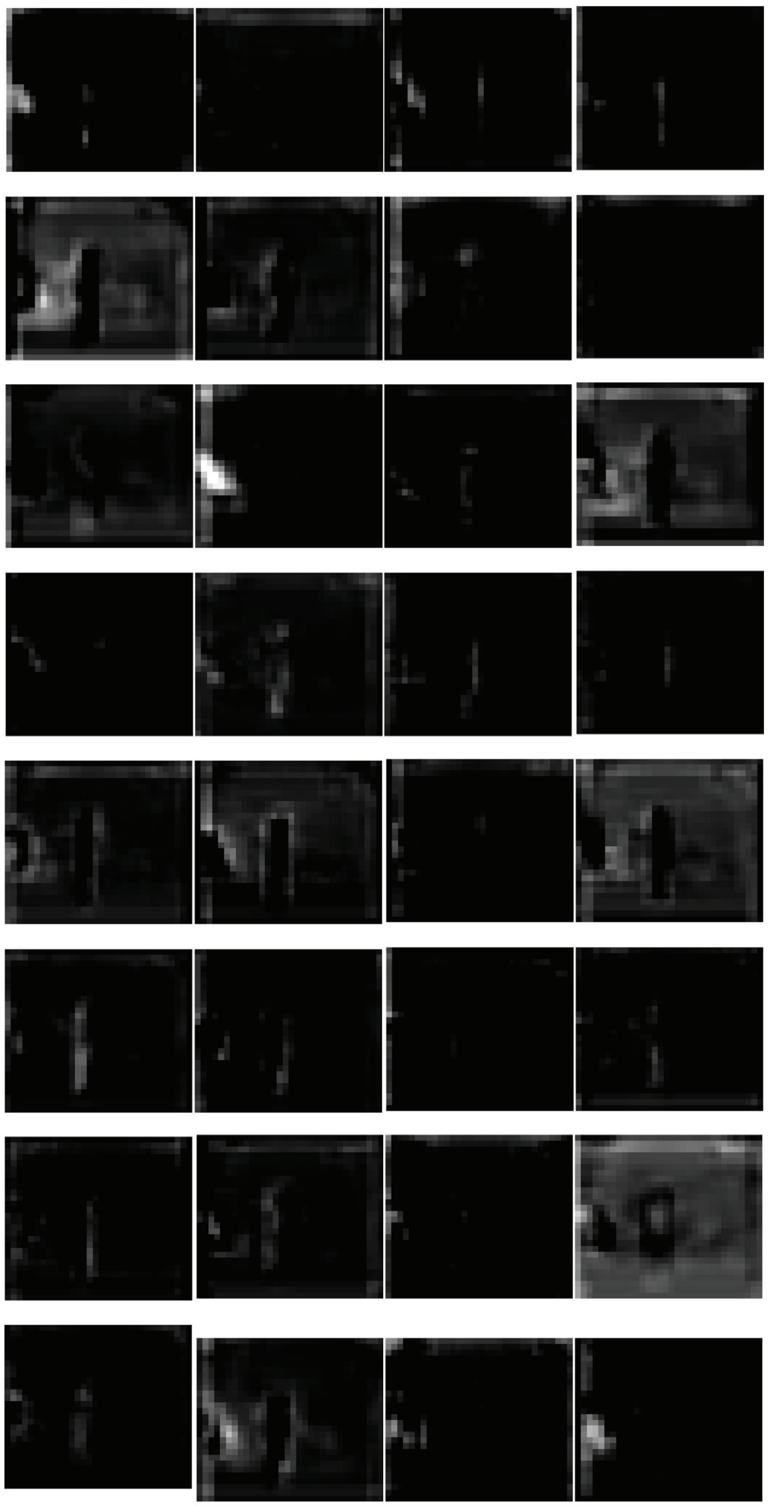


Figure B.11. Layer 8 activations (part 1)

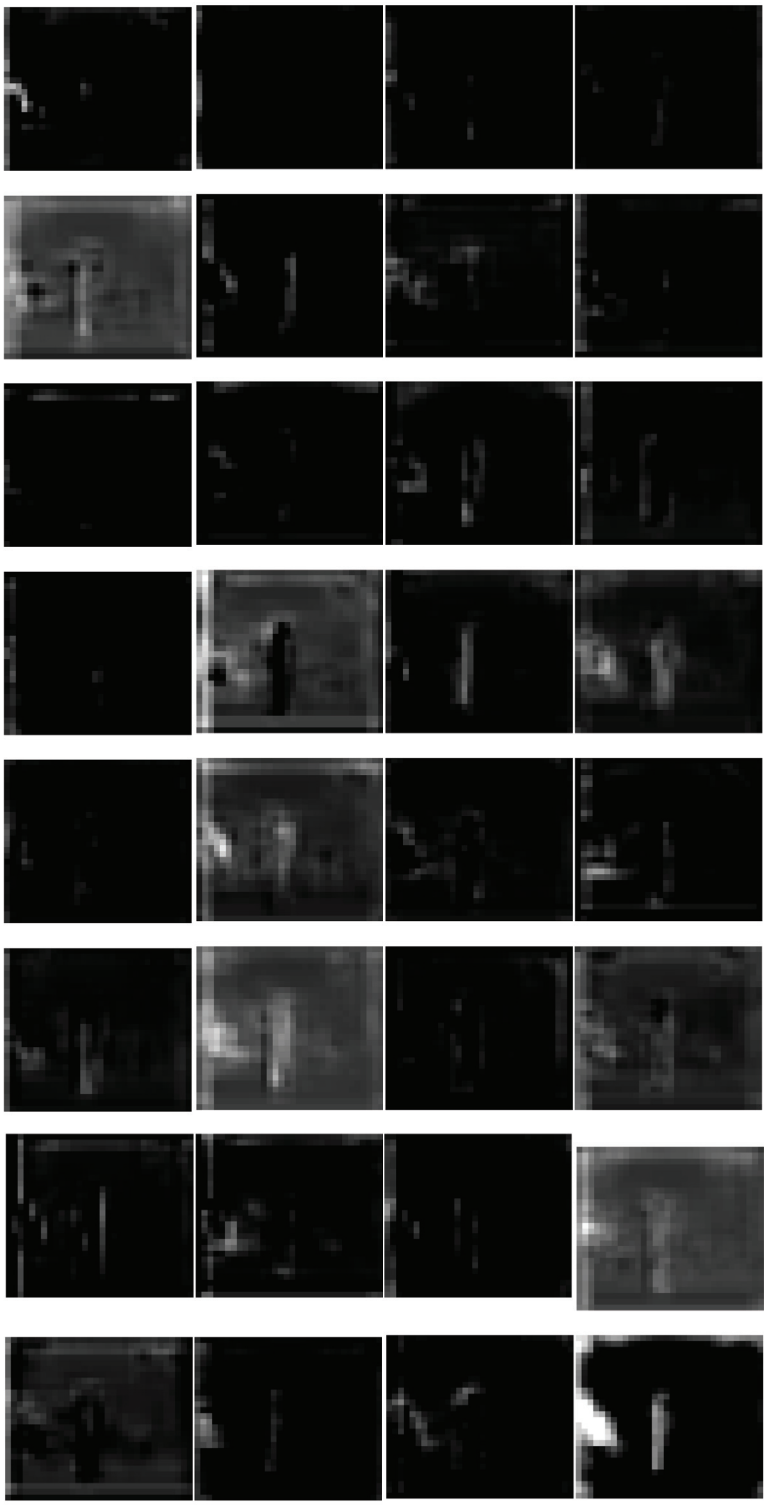


Figure B.12. Layer 8 activations (part 2)

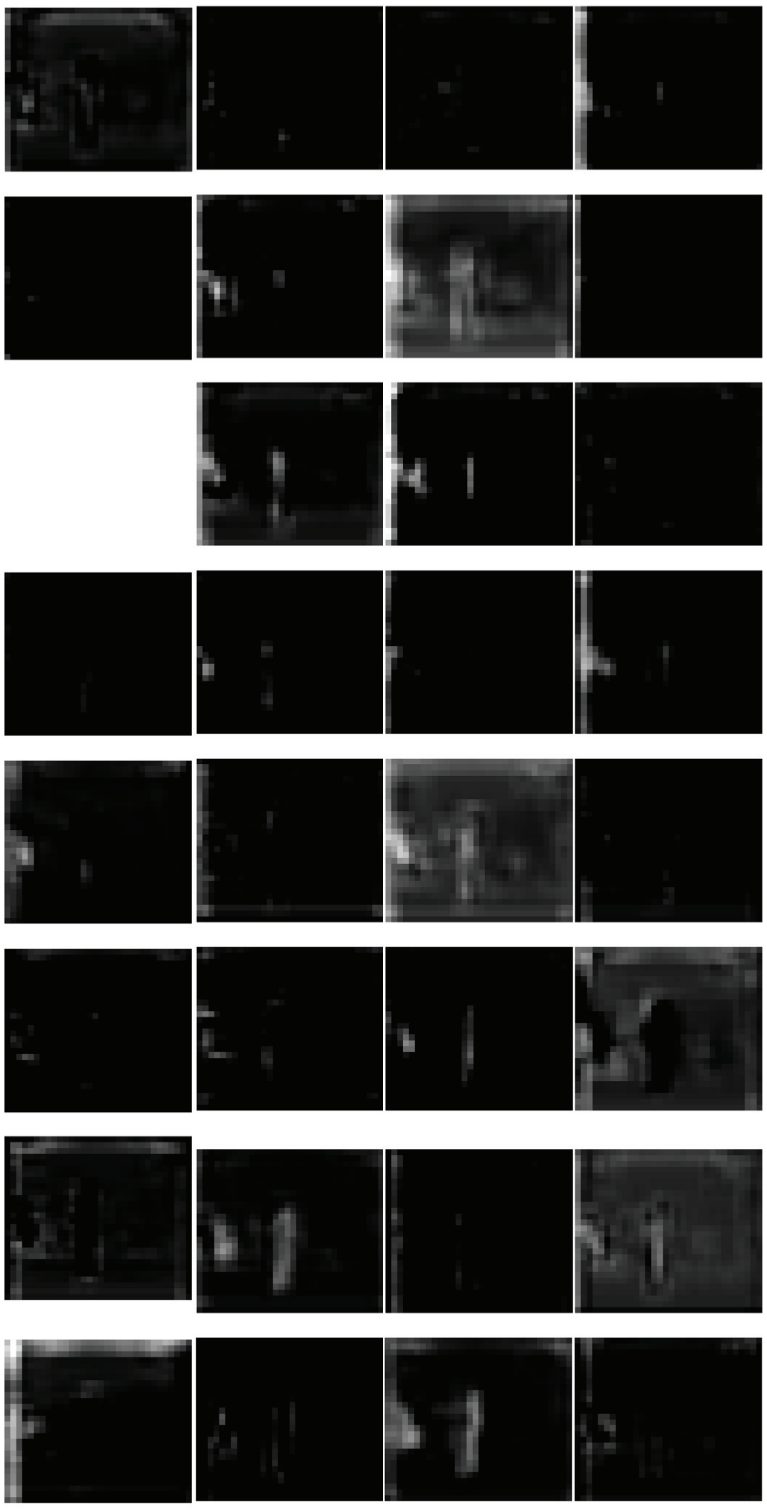


Figure B.13. Layer 8 activations (part 3)

Figure B.14. Layer 8 activations (part 4)

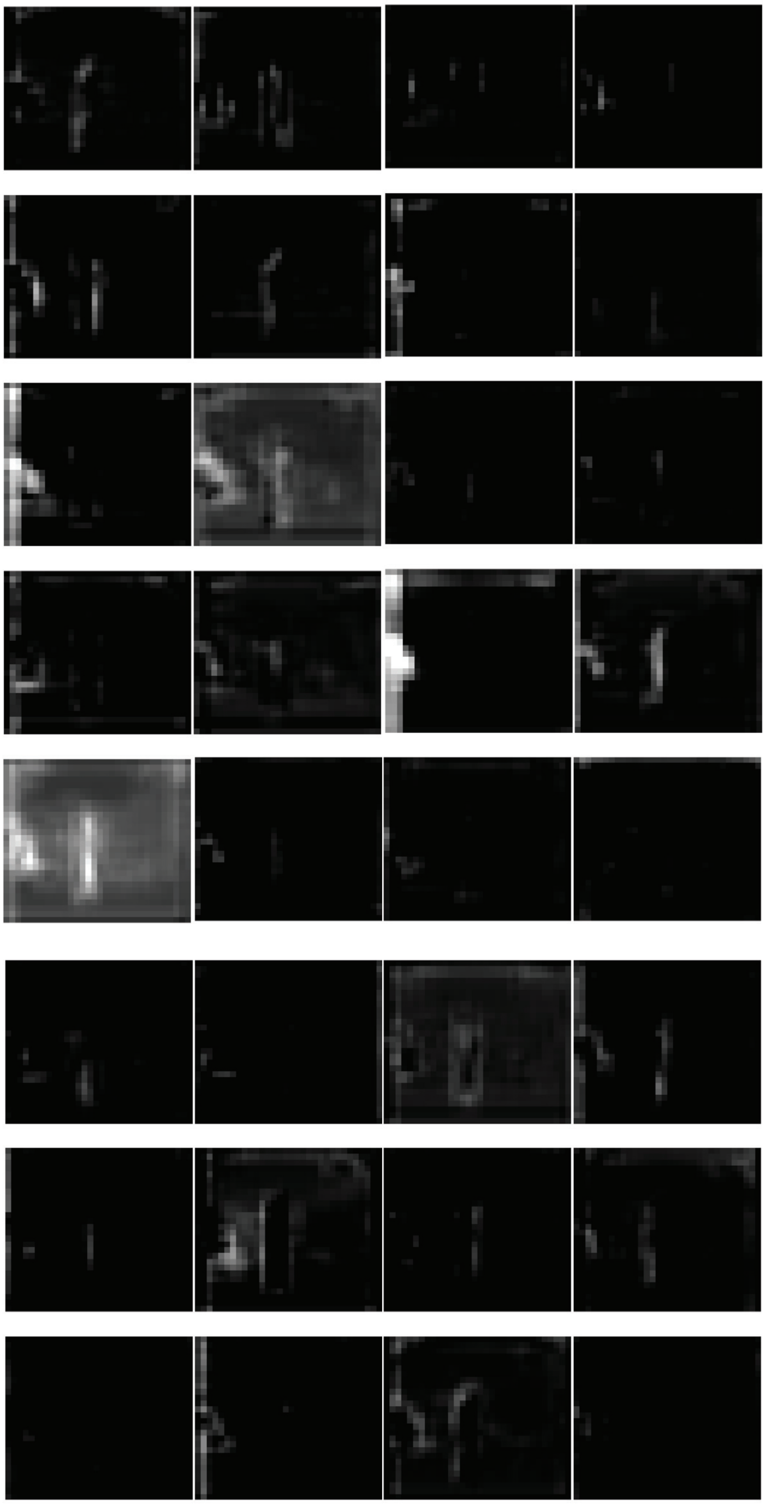


Figure B.15. Layer 8 activations (part 5)

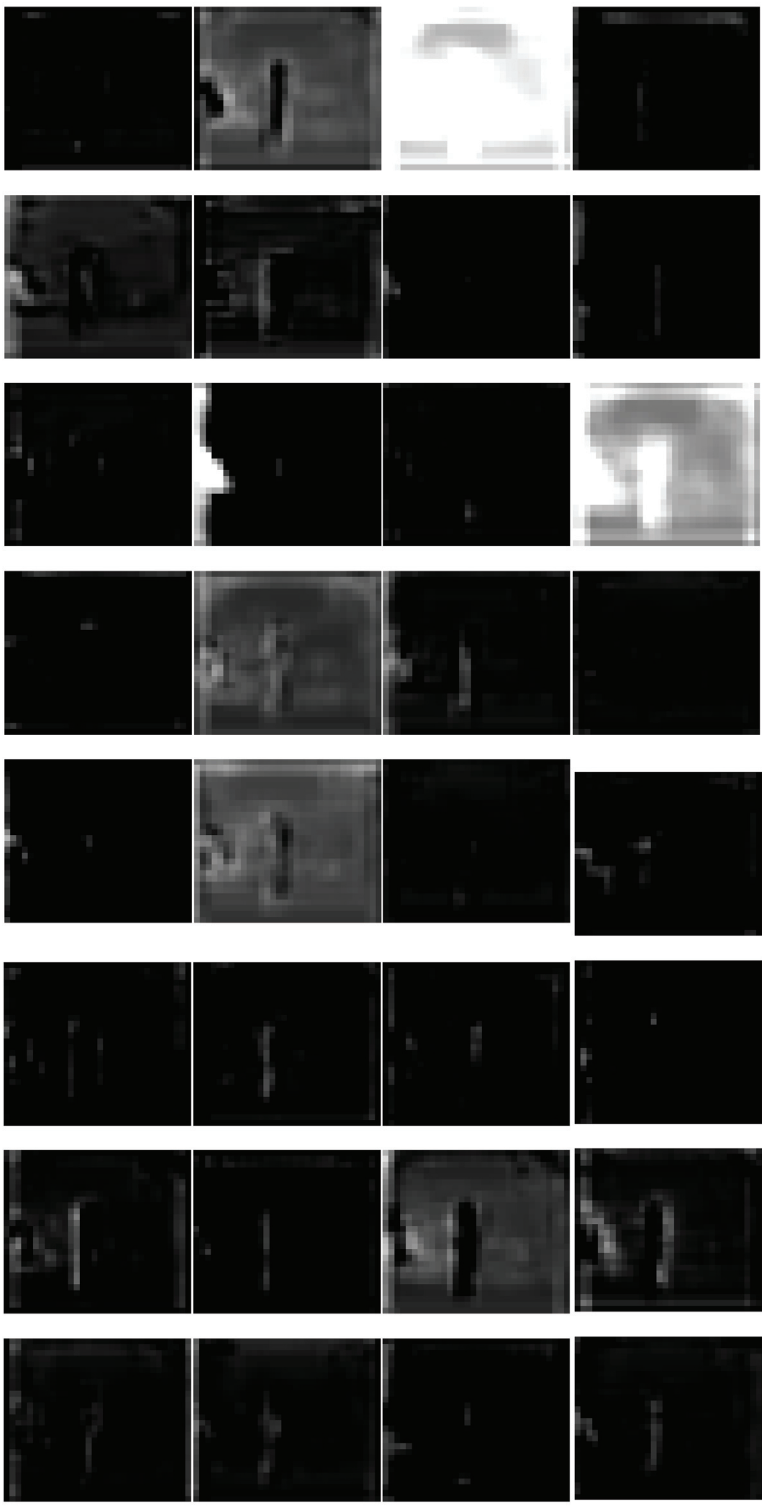


Figure B.16. Layer 8 activations (part 6)

Figure B.17. Layer 8 activations (part 7)

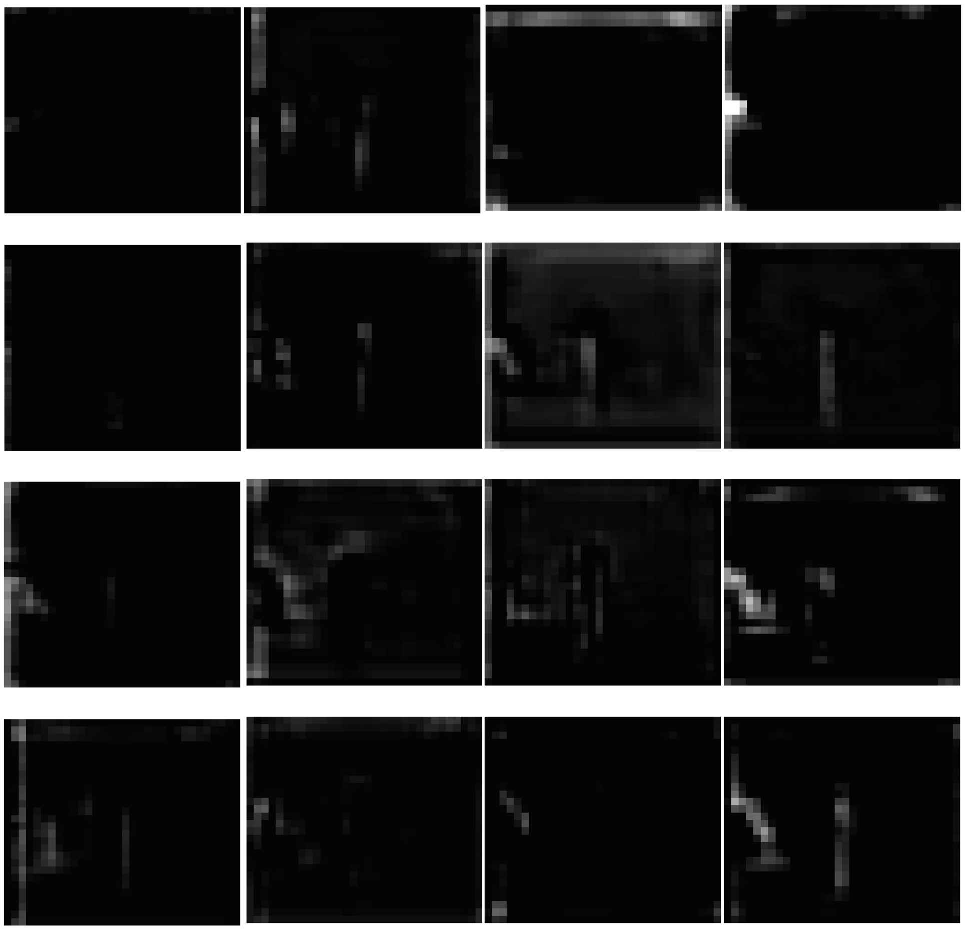


Figure B.18. Layer 8 activations (part 8)

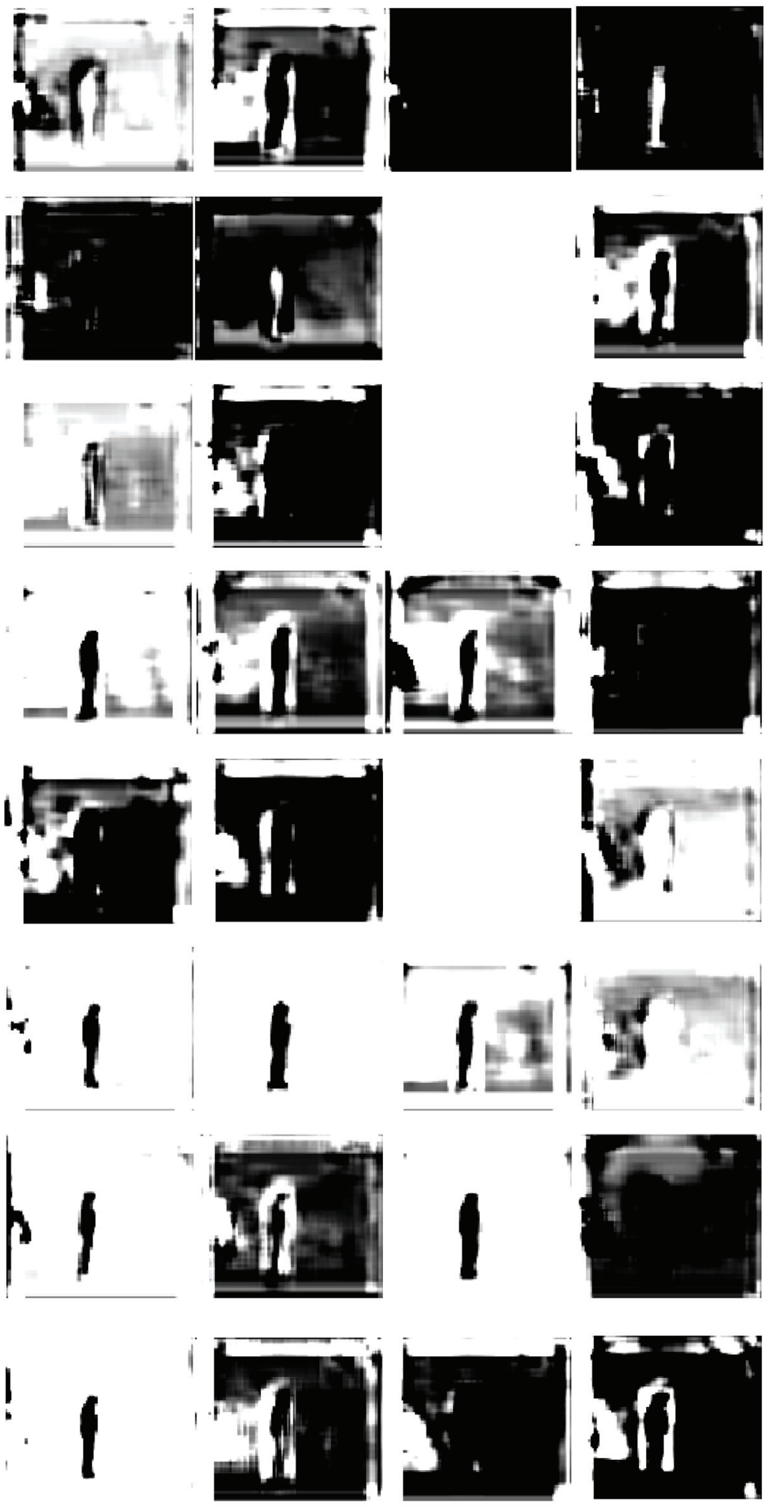


Figure B.19. Layer 10 activations (part 1)

Figure B.20. Layer 10 activations (part 2)

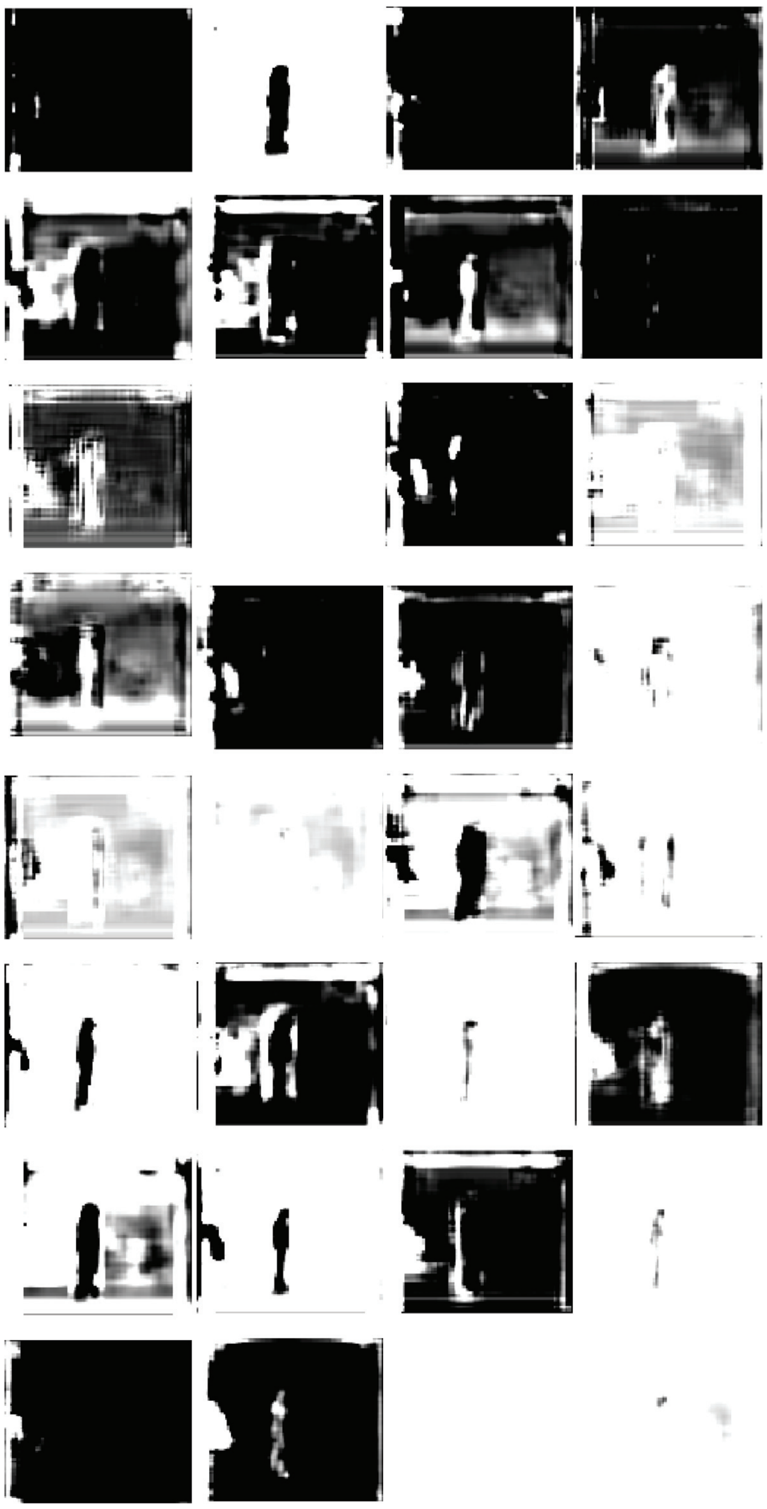


Figure B.21. Layer 10 activations (part 3)

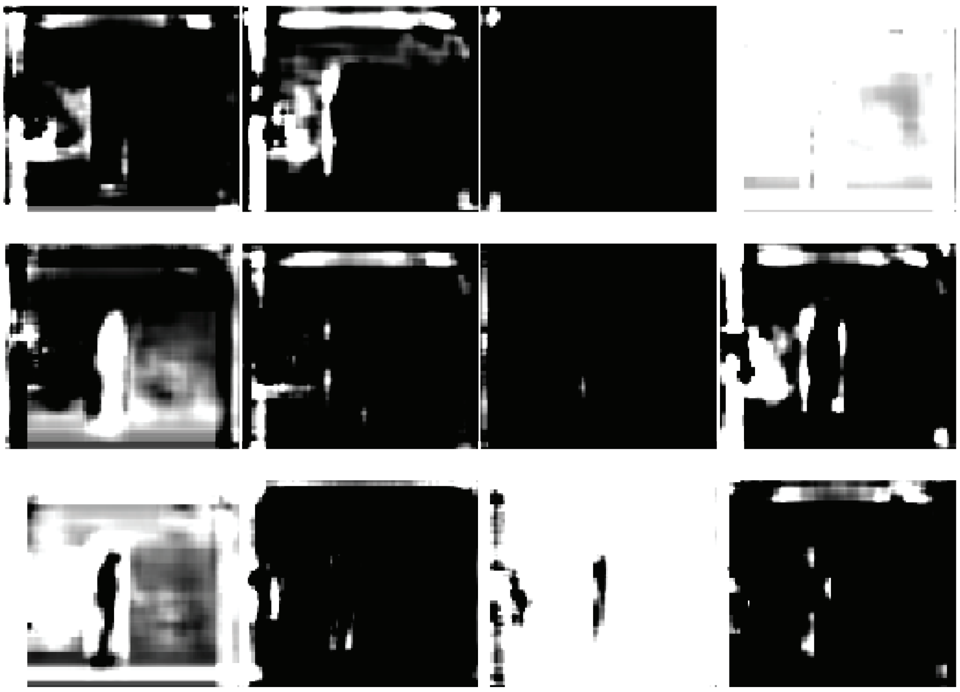


Figure B.22. Layer 10 activations (part 4)

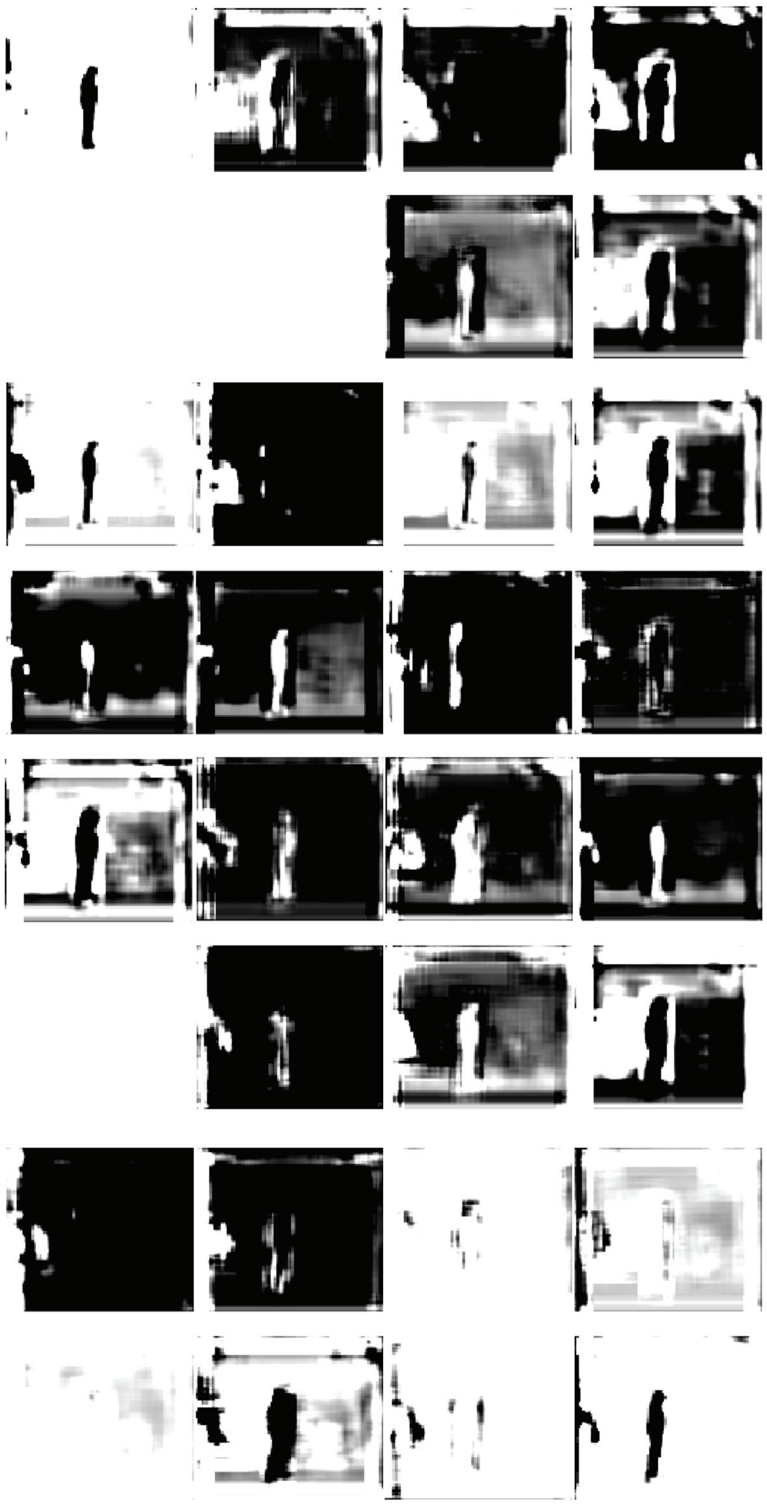


Figure B.23. Layer 11 activations (part 1)

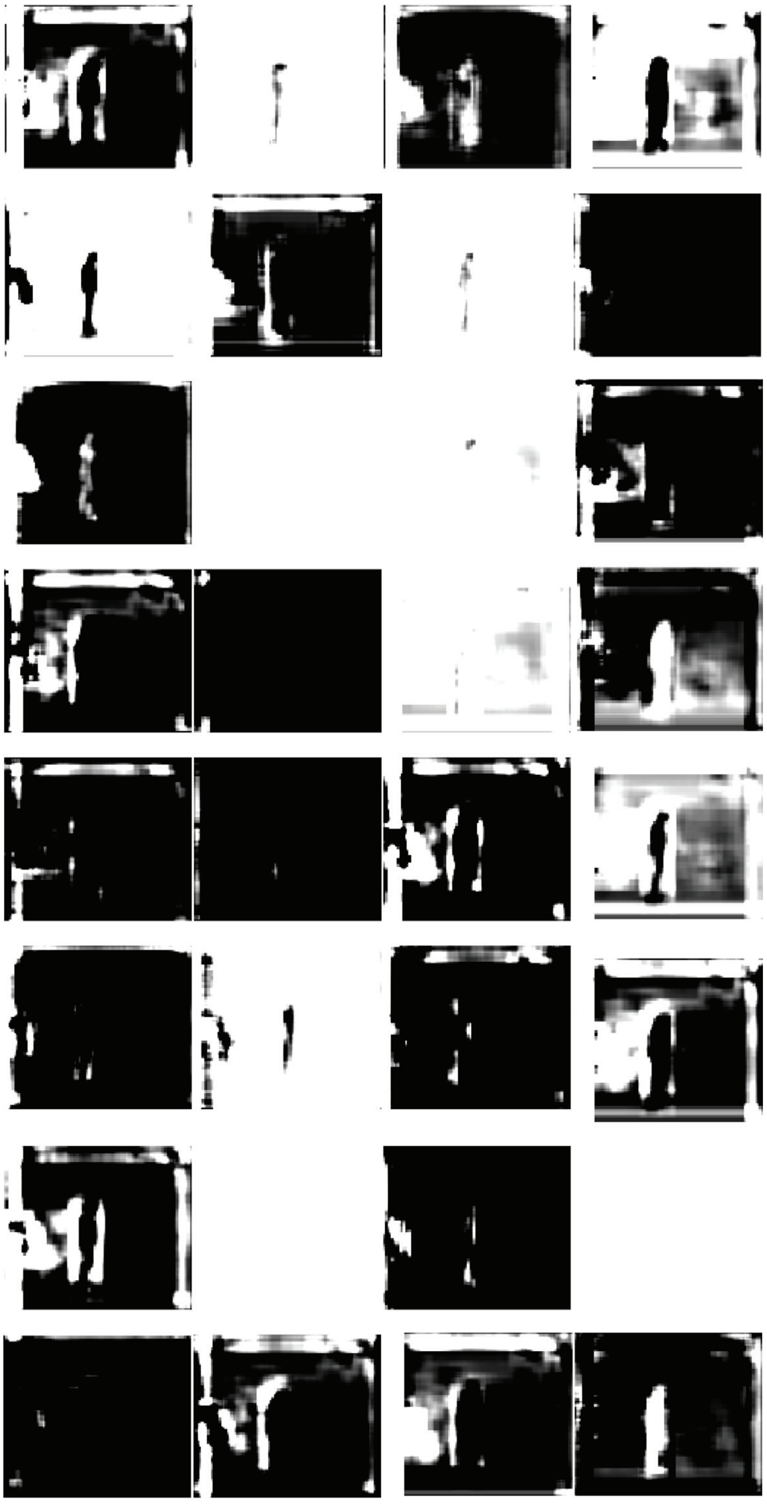


Figure B.24. Layer 11 activations (part 2)

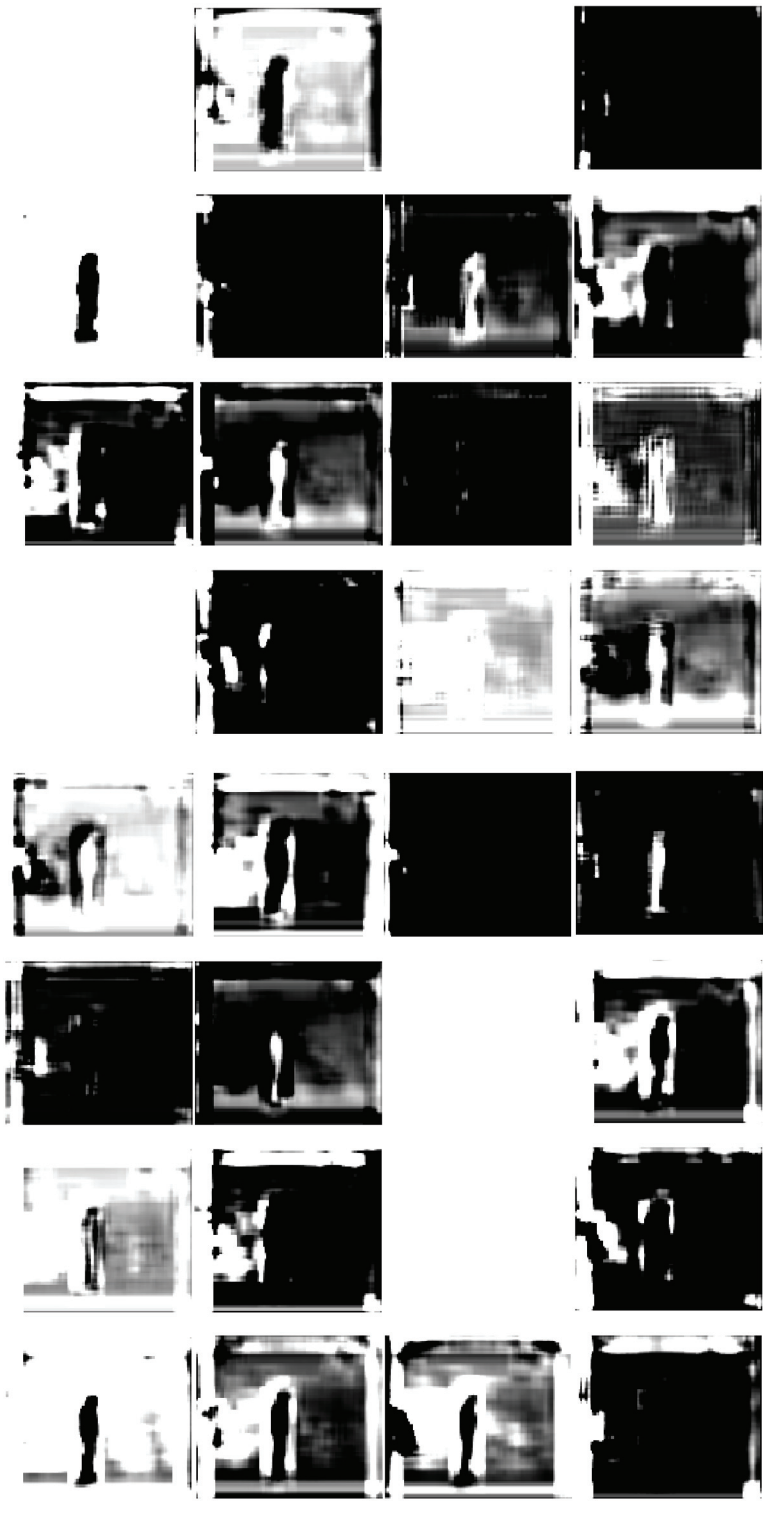


Figure B.25. Layer 11 activations (part 3)

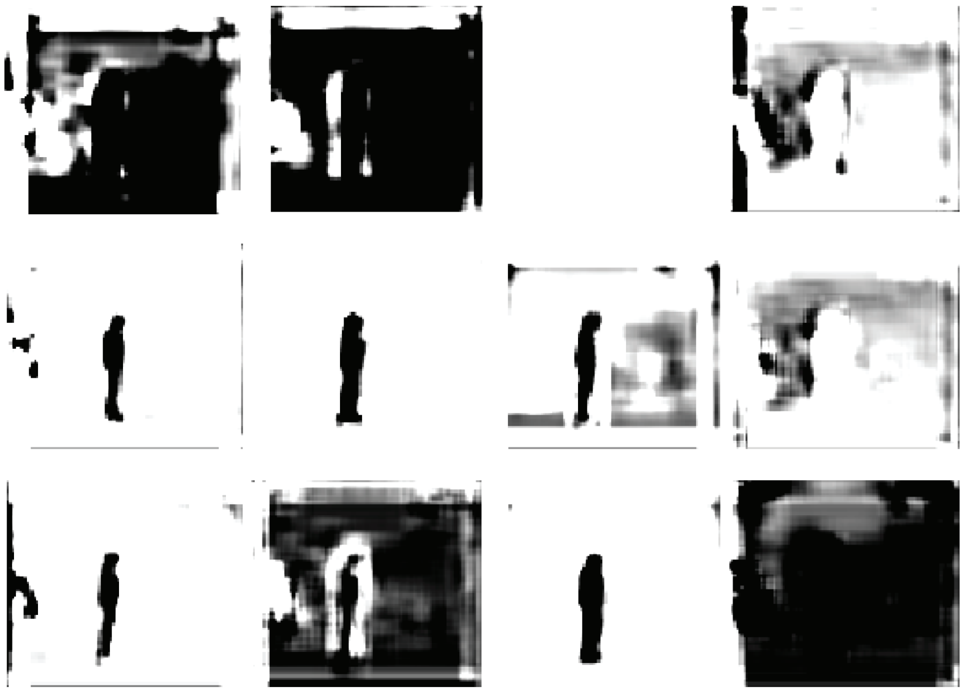


Figure B.26. Layer 11 activations (part 4)

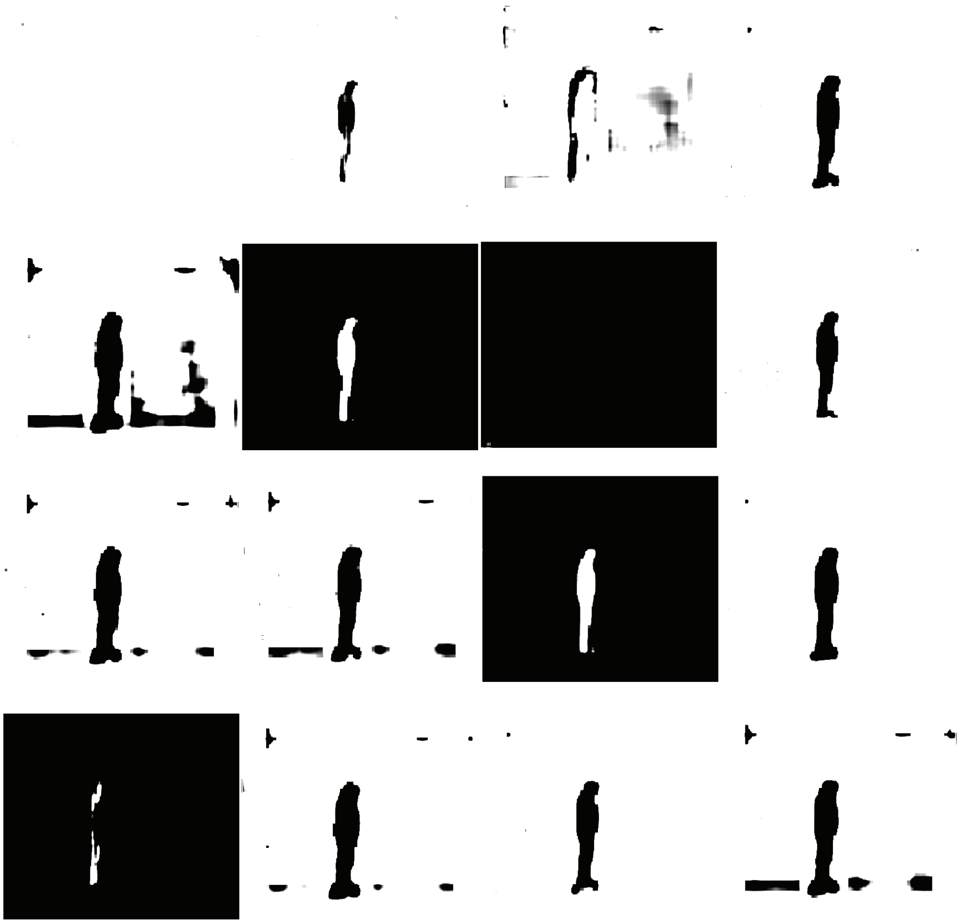


Figure B.27. Layer 12 activations

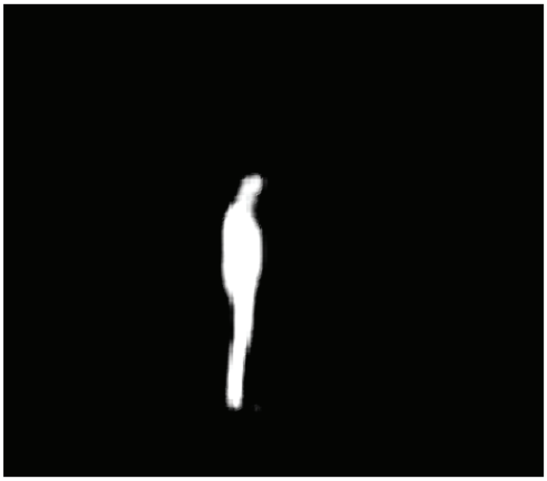
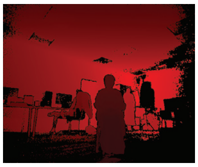
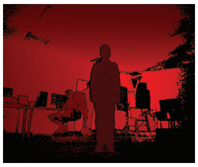


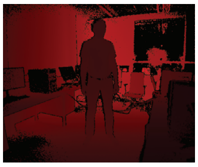
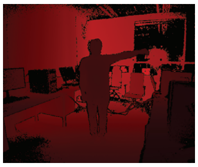
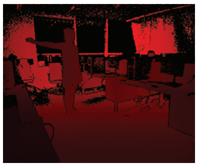
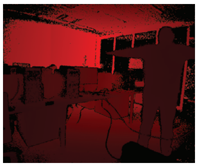
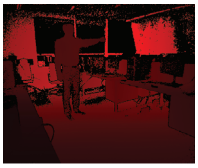
Figure B.28. Layer 13 activation (output image)

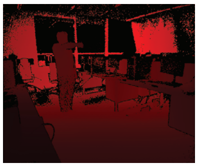
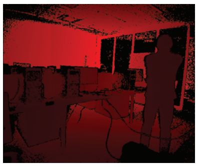
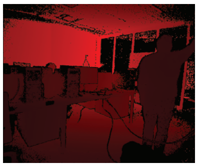
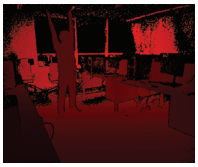
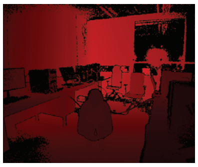
C. LIST OF POSES CAPTURED IN DATASETS

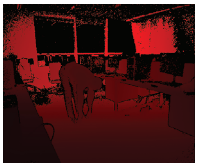
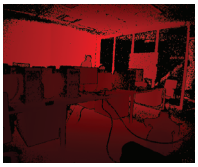
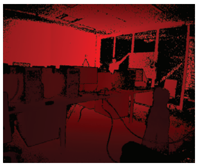
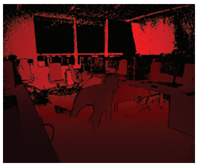
Simple dataset

	Sitting on a chair
	Standing

Complex dataset

	Standing relaxed
	Standing with right hand lifted half way
	Standing with left hand lifted half way
	Standing with both hands lifted half way
	Standing with right hand put forward

	Standing with left hand put forward
	Standing with both hands put forward
	Standing with left hand lifted
	Standing with right hand lifted
	Standing with both hands lifted
	Standing with right arm at 90 degrees angle
	Standing, side view
	Squatting, hands holding legs

	Bent forward, hands reaching floor
	Laying on the ground
	Bent forward, one leg is raised and put backwards
	Laying on the ground, legs up
	Standing, bending backwards with raised hands
	Low start position

C.1. Full software solution for semi-automatic segmentation

The full software solution has all analyzed algorithms and performance improvements implemented. The user may choose whichever settings he sees fit – the segmentation algorithm (bounding box, radius search, K-Means cuts), segmentation sensitivity, add segments manually or removes segmentation output by using an eraser. Most importantly, the segmentation output is shown to the user on the screen. The full set of features is presented in the UML use case diagram in Figure C.1.

A screenshot of the solution is shown in Figure C.2 to better visualize the tool and how it is supposed to help with segmentation. The top of the screen is filled with option selection inputs and action buttons. 12 images are shown on the screen (the screen resolution was 2560x1440 pixels). Different shades of gray represent different depth values. The user then identifies the location of the human and segments it by using the selected method. If one of the automatic algorithms is selected, the user has

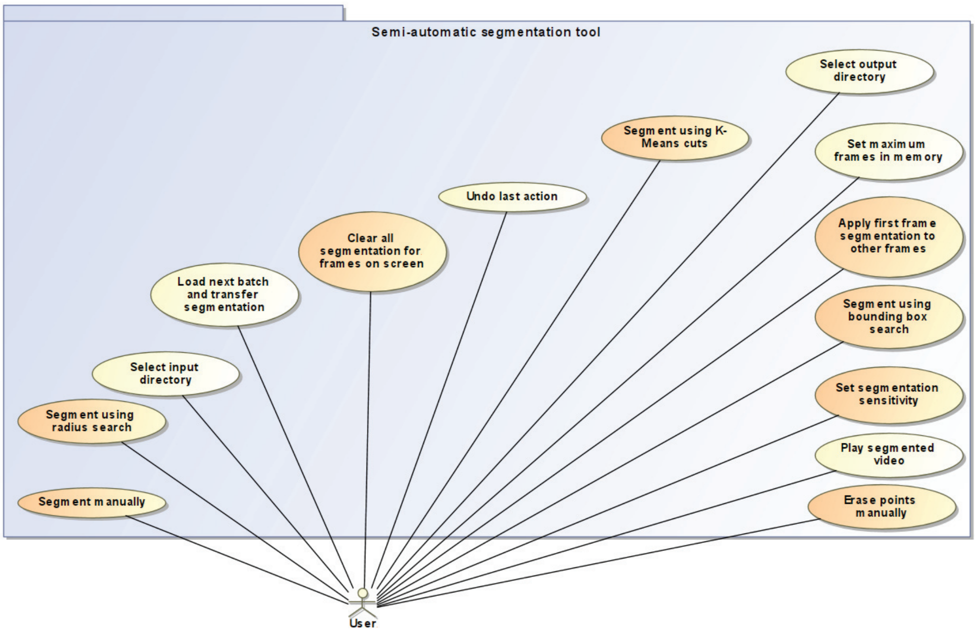
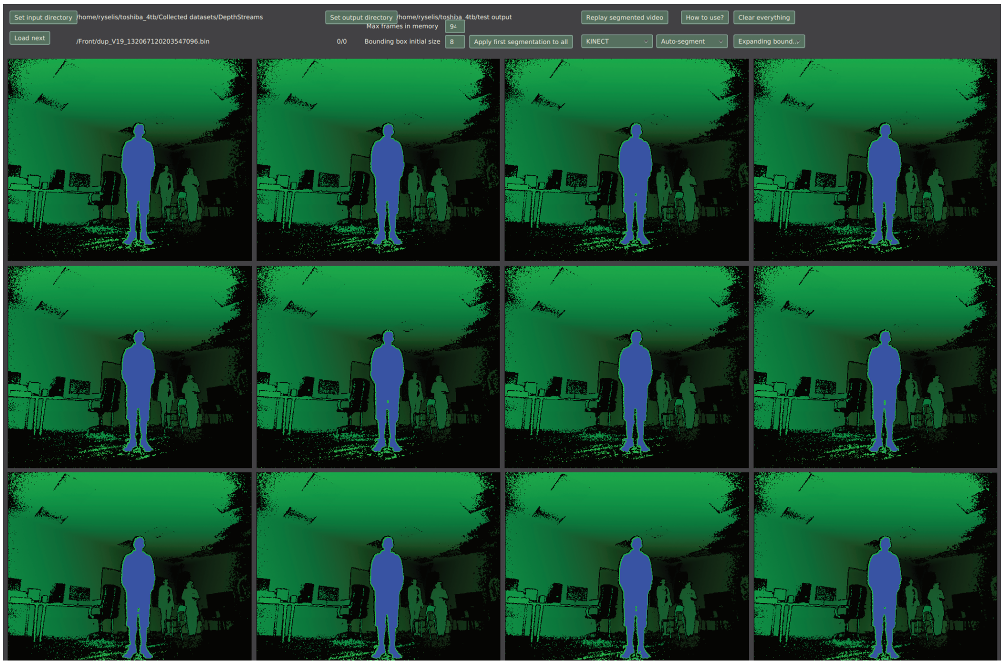

Semi-automatic segmentation tool
Select output directory
Segment using K-Means cuts
Set maximum frames in memory
Undo last action
Apply first frame segmentation to other frames
Clear all segmentation for frames on screen
Load next batch and transfer segmentation
Segment using bounding box search
Select input directory
Set segmentation sensitivity
Segment using radius search
Play segmented video
Segment manually
Erase points manually
User

Figure C.1. Use case diagram for semi-automatic segmentation tool



/home/ryselis/toshiba_4tb/Collected datasets/DepthStreams

/Front/dup_V19_132067120203547096.bin

/home/ryselis/toshiba_4tb/test output

Max frames in memory 94

0/0 Bounding box initial size 8

KINECT · Auto-segment · Expanding bound...

Figure C.2. Screenshot of the software solution

to click on one of the pixels of the human in the scene and this point is used as the initial point for the segmentation. The screenshot shows the state after automatic bounding box segmentation and clicking the button *Apply first segmentation to all* – all the frames were segmented by using the same initial point. As a result, a total of two clicks were required to segment 12 frames. Clicking *Load next* will apply the same segmentation to the next 12 frames. It also works for this particular instance, so 24 frames are selected by using 3 clicks. It is also easy to visually confirm that the segmentation is correct. It only takes about one second for a trained individual to verify that the segmentation is correct. This, of course, is the best case scenario, but a large portion of the dataset can be segmented this way as shown in Section 4.2.5.1.

UDK 004.5+004.422.635.33+004.032.26](043.3)

SL 344. 2023-05-08, 29,5leidyb. apsk. l. Tiražas 14 egz. Užsakymas 74.
Išleido Kauno technologijos universitetas, K. Donelaičio g. 73, 44249 Kaunas
Spausdino leidyklos „Technologija“ spaustuvė, Studentų g. 54, 51424 Kaunas