Tartu University

Faculty of Science and Technology

Institute of Technology

Jüri Jõul

3D bounding box detection of moving objects for robot navigation in dynamic environments

Bachelor's thesis (12 EAP)
Computer Engineering

Supervisor:

Arun Kumar Singh

Tartu 2021

Resümee/Abstract

Liikuvate objektide 3D-piirikastide tuvastamine robotnavigatsiooniks dünaamilistes keskkondades

Liikuvate objektidega kokkupõrgete vältimiseks kasutatakse robootikas laialdaselt kolmemõõtmelist objektituvastust. Tuvastatud objektide trajektooridest ja tuvastussüsteemi mürast lähtuvalt saavad robotid kasutada teekonna planeerimise algoritme, navigeerimaks rahvarohketes keskkondades. Selles bakalaureusetöös pakutakse välja hübriidne lähenemine 3D-objektituvastuseks, ühendades sügavusandmed kõrgelt arenenud 2D-objektituvastussüsteemiga. Töö toimimist demonstreeriti simuleeritud ja pärismaailmas läbi viidud eksperimentidega. Ühtlasi teostati tuvastussüsteemi kohta põhjalik müraanalüüs, mida on võimalik kasutada tulevikus robotnavigatsiooniks määramatust sisaldavates olukordades.

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia; T125 Automatiseerimine, robootika, control engineering; T111 Pilditehnika

Märksõnad: objektituvastus, robootika, piirikast, algoritmid, ROS

3D bounding box detection of moving objects for robot navigation in dynamic environments

3D object detection is widely used in the field of robotics to avoid collisions with dynamic objects, such as humans. Based on the trajectories of the detected objects and the noise of the detection pipeline, robots can use motion planning algorithms to safely navigate in crowded environments. This thesis proposes a hybrid approach to 3D object detection using depth data fused with a mature 2D object detector. Both simulated and real-world experiments were performed as a demonstration of the solution. An extensive noise analysis of the developed detection pipeline was also carried out for future use in robot navigation under uncertainty.

CERCS: T120 Systems engineering, computer technology; T125 Automation, robotics, control engineering; T111 Imaging, image processing

Keywords: object detection, robotics, bounding box, algorithms, ROS

Contents

List of Figures

Abbreviations, Constants, Generic Terms

Bounding box - the minimal box enclosing an object

CNN - Convolutional Neural Network, a commonly used neural network in computer vision

Coordinate frame - a system that uses one or more numbers, or coordinates, to uniquely determine the position of a point

DNN - Deep Neural Network

FPS - Frames Per Second, framerate

GPU - Graphics Processing Unit

Gazebo - a robot simulator compatible with ROS

LiDAR - Light Detection And Ranging

Object detection - the process of identifying an object and its coordinates

Point-cloud - a set of data points in space, a common way to represent 3D data; conversion between RGB-D and point-cloud data is possible

R-CNN - Region-based Convolutional Neural Network

RGB - Red, Green, and Blue (a regular color image)

RGB-D - Red, Green, Blue, and Depth (a color image with an added depth channel)

RMS - Root Mean Square

ROS - Robot Operating System

RViz - a tool used for visualization in robotics

Stereo camera - a camera with two lenses and sensor, simulates human binocular vision; can capture depth information about the image; synonymous with depth camera

YOLO - You Only Look Once, a 2D object detector

1 Introduction

Motivation: There are several applications where a mobile robot needs to navigate amongst humans in indoor spaces such as hospitals, malls, office-buildings etc [1, 2, 3, 4]. A critical component of any robotic software is the motion planning algorithms that ensure that the mobile robot can navigate safely without colliding with the humans. The motion planning algorithms needs obstacle (human) detection as an input. The main objective of this thesis is to create such a pipeline. Specifically, the thesis develops the software to detect humans through an RGB-D camera and then create 3D bounding boxes (or cylinders) around them. These 3D bounding boxes can then be used by any off-the-shelf motion planning algorithms.

1.1 Problem Overview

The problem addressed in this thesis can be formally described in the following manner.

Definition 1 *Given an RGB-D image from a camera mounted on the robot, detect humans in the scene and construct a 3D bounding box around it. The center of the bounding box should also be known in either the robot's local or world frame. Furthermore, if the scene has multiple humans, then separate bounding boxes need to be created for each of them along with a unique ID for tracking the humans as the robot moves.*

Based on the problem definition, the proposed thesis falls into the ambit of object detection that has been extensively studied in computer vision literature. Object detection in the context of computer vision is the process of determining the positions and classes of objects from sensor data [5]. The detection output is generally in the form of bounding rectangles in 2D or bounding boxes in 3D, although the term bounding box has come to be used for both. A bounding box is defined as a box in 3D space that encompasses all points of a given object. Since a bounding box can conveniently encode an object's position and orientation, it has become the *de facto* way to describe 3D objects in the context of 3D object detection [5].

Although a subject of vast research, object detection as a problem has not been solved. Lately, as the computational capacity of computers has increased, especially in the form of parallelized computation performed in the GPU, researchers have focused their efforts on artificial neural network based approaches [6, 7, 8, 9, 10]. Artificial neural networks are a form of machine learning which mimic the structure of the neural networks found in biological brains [6]. A subset of neural network based machine learning, called deep learning, has been the basis of many advances in the field of computer vision [11].

1.2 Thesis Organization

This thesis is divided into five main parts. The current chapter gives an introduction to the problem. Chapter 2 gives an overview of the state of the art approaches in the field of 3D object detection. In Chapter 3, the developed solution is explained in detail, while providing the necessary background information. Chapter 4 shows simulation results (including a real-world experiment) and Chapter 5 presents an extensive noise analysis of the developed detection pipeline. The final chapters conclude the work and give a short overview of possible future work.

2 State of the Art

Bounding box detection from 3D point-cloud data has been an active area of research in both computer vision and robotics community because of its immense utility in autonomous navigation/driving. In this chapter, these approaches are reviewed and contrasted with the proposed method of this thesis. The review is split into two parts: (i): purely data-driven/learning based approach that is predominant in the computer vision community and (ii) approaches used in robotics that rely more heavily on the classical image processing techniques.

2.1 Data-Driven Approaches

There have been many approaches to data-driven 3D object detection. Researchers have to consider whether to use 2D images, 3D point-clouds or both as inputs to their systems. Mature 2D object detectors have been in use for a long time, enabling them to be utilized as part of modern 3D object detectors to improve accuracy and efficiency, allowing for real-time inference [9]. Native 3D detectors have only recently been seriously explored, thanks to the increase in computing power, enabling it to be also used in real-time applications. Still, a hardware accelerator or a powerful GPU is often needed for both, at least in order to enable real-time detection needed for real robots in dynamic environments [5]. Considering the many possible combinations of input data and types of commonly used detectors, there are a multitude of ideas and solutions which have already been examined. A brief overview is given of the state of the art based on each of the aforementioned approaches.

Starting with monocular images, there have been interesting ideas which have shown good results. Marcel Cata Villa uses both neural network based 2D and 3D detectors, first of which is the commonly used Faster R-CNN detector and the second which comprises three parts: location, dimensions, orientation, all of which firstly assign detections to bins and then regresses the prediction further. The resulting detector was successfully benchmarked [12]. Although the run-time has not been discussed, the use of Faster R-CNN limits the detector to about 5 FPS in the best case [13]. Ma et al. have also pushed forward the state of the art of monocular approach with their work. They took the approach of representing the 2D image as a 3D point-cloud and then use the RGB data as an augmentation to the 3D point-cloud. Again, the computational complexity and run-time performance is not analysed [14].

The other usual approach to object detection is using LiDAR. Lin Yan, Kai Liu, Evgeny Belyaev and Meiyu Duan have proposed a real-time 3D detector based on LiDAR input, which applies sparse convolution on point-clouds and makes predictions using a dense detection network in 2D. Due to the exploitation of sparsity of the point-cloud data, they were able to achieve a detection speed of 40 FPS and their model also shows good performance in the benchmarks [10]. Ali et al. modified the YOLOv2 2D detector to also include regression for the yaw angle, 3D box

center and the height of the box. They achieved real-time performance with 40 FPS and adequate benchmark scores on the KITTI dataset [15].

Although LiDARs usually have superior long-range accuracy, depth cameras are often considerably more affordable and with a more compact form factor, allowing them to be used in smaller robot platforms. In addition, depth cameras can have better accuracy in shorter range [16]. Due to these constraints, depth cameras have often been considered as a solid alternative. Qi et al. use a fused approach with RGB-D data. Their network generates 2D region proposals using a CNN and predicts 3D bounding boxes based on the the 3D frustums of proposed regions. They achieved state of the art results in different benchmarks with the real-time performance of 5 FPS [9].

2.2 Classical Approaches

Recently, research has mostly converged towards exploring the deep learning side of machine learning and traditional algorithms are often overlooked. Some of the traditional approaches to 2D object detection include Haar feature-based cascades, Scale-Invariant Feature Transforms (SIFT) and Histogram of Oriented Gradients (HOG) based detectors [17, 18, 19]. These can be used as a basis for 3D object detection but many novel approaches using none of the traditional algorithms have also been proposed, including algorithms operating directly on the depth data.

Lin et al. [20] proposed a solution using depth value histograms called U-depth maps to detect three-dimensional bounding boxes of any objects. They have measured their average position and velocity estimation errors and also the performance of their approach. With their respective errors being around 0.27 m and 0.44 m, and the achieved detection period being around 8 ms, it can be considered a reasonable result in terms of frame rate vs accuracy [20]. Adarsh Jagan Sathyamoorthy et al. have proposed a similar approach to that used in this thesis [21]. That is, it consists of first estimating the 2D bounding box from an RGB image and then overlaying it on the depth map. However, they do not elaborate their choice of 2D descriptors and furthermore did not perform any noise analysis on the measurement [21].

2.3 Contributions

The approach presented in this thesis tries to fuse the best of both the data-driven and classical image processing based approaches. YOLOv5 [7] is used to detect 2D bounding boxes in monocular images. It is then used as a basis to filter the depth map of the scene and extract depth information of detected objects. The depth map used is generated by the Intel RealSense D435i stereo camera. There are two key advantages to this approach. First, it does not rely on the annotated data of 3D bounding boxes from depth images. These data are difficult to obtain for a given custom application (e.g navigation of robots in hospitals). The available public data sets predominantly cover outdoor autonomous driving applications. On the other hand, it also does not rely purely on classical approaches and instead uses a pre-trained 2D object detector to efficiently guide its 3D detection algorithm. Another key contribution of this thesis lies in the extensive noise analysis to characterize the distribution of the error in the detection pipeline. As shown later, the error distribution is non-trivial and shows substantial deviation from the standard Gaussian assumption. The thesis also provides preliminary results on tracking different humans in a dynamic scene.

3 Detection Pipeline

One of the main contributions of this thesis was creating a working solution of a 3D object detection pipeline as a ROS package. In this chapter the choices and implementation details of the creation of this pipeline are explained and analysed, starting from choosing the sensor inputs and ending with the outputted 3D bounding box. All the software developed for this thesis was documented and hosted publicly on GitHub [22].

3.1 Overview

Fig. 3.1 shows the overview of the detection pipeline. It consists of a depth camera which provides the RGB-D image (point-cloud) of the environment to the detection pipeline. The RGB image is also fed to the YOLOv5 neural-network block that uses it to compute a 2D bounding box for any object present in the scene. The output of the YOLOv5 detector is in turn fed back into the detection pipeline, where it is merged with the depth map to estimate 3D bounding boxes based on the 2D detections. Before the actual detection, an additional step of converting the images to correct formats is needed for the detection algorithm. Predicted 3D bounding boxes pass on to the tracking algorithm which in turn creates the inputs for the trajectory prediction algorithms. To view the detection, tracking and trajectory prediction results, a visualization tool RViz can be used. Although not used under normal operation, a custom viewer application was also developed for easier annotation of image data.

3.2 Camera

The assortment of different sensors for data acquisition for computer vision is vast. The most commonly used sensors include regular color cameras, stereo cameras for depth information and LiDARs. Each of them bring both benefits and disadvantages to the robotics researcher. In the following section, a thought-process of choosing the camera to be used is laid out, followed by the overview of the integration of the camera into the pipeline. Relevant background information about the camera itself and the interface software is also given.

3.2.1 Choice of Camera

Regular 2D cameras do not have any inherent ways of acquiring depth information about the scene, although they are very inexpensive and easy to use. Stereo cameras are more expensive than regular cameras but provide the huge advantage of gaining a depth channel aside from the red, green and blue color channels of the regular camera. LiDARs are even more expensive but provide a detailed depth map of the entire scene. Aside from the cost, their main disadvantages lie in their sub-optimal performance in the close range and their larger form factor.

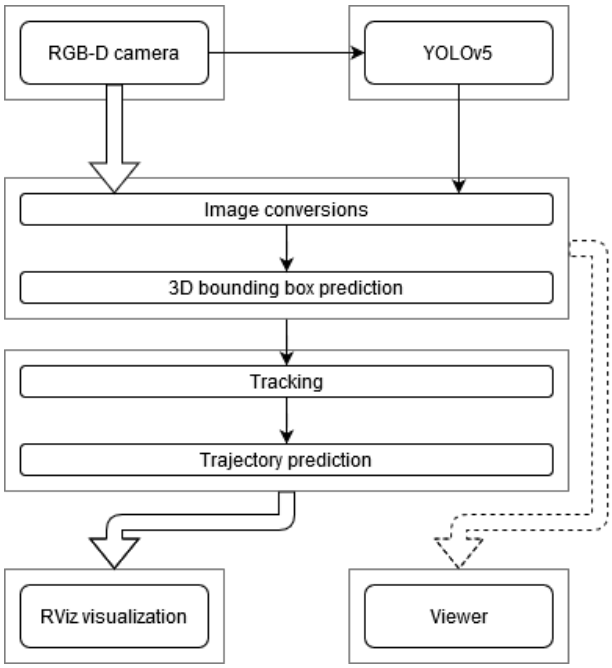


Figure 3.1: Overview of the detection pipeline with tracking, prediction and visualization

Due to the interest of using the results of this thesis on mobile robots and even drones, a small form factor lightweight sensor was needed. A regular camera's depth accuracy was considered subpar and a LiDAR too large, heavy and expensive. Therefore, for this thesis, the Intel RealSense D435i Depth Camera was used.

The D435i features an RGB camera and a stereo pair of IR cameras for depth sensing. An IR generator is used for increasing the depth accuracy. Although not relevant to this thesis, the camera also features an inertial measurement unit [23]. The RealSense cameras are widely used by the industry, as they come with a free Software Development Kit (SDK) and a wrapper for ROS. The ROS wrapper is essential for this thesis, as the whole pipeline is developed as a ROS package and in keeping with its ethos, requires camera feed to be received using ROS topics. In addition to the free software, there are various published whitepapers and guidelines for using this camera, which makes the development process much easier.

3.2.2 RealSense and ROS Integration

ROS is a Linux-based open-source platform for developing modular robot software [24]. It comprises various tools, libraries and standards meant to assist in the robot software development process. The way ROS is structured is ideal for collaboration among robot developers, as the software is packaged to be reusable. Another feature of ROS is the distributed nature of its component programs - nodes, which allows different parts of a robot's system to run on different computers, requiring only a network connection [24]. In this chapter, the focus is on the integration of ROS and the depth camera.

As discussed in the previous paragraph, ROS software typically comprises different nodes. These nodes are essentially separate programs which can communicate with each other using standardized ROS messages. These messages are broadcasted to and listened from ROS topics. Important ROS-related tools include the Gazebo simulator and RViz (ROS Visualization) visualization software [25, 26]. The former was used to simulate a robot in a human environment and the

latter was used to visualize the data the robot received.

ROS has many different versions and a choice was made to use ROS Melodic for this thesis. This was in part due to a Clearpath Robotics robot Jackal used in the Collaborative Robotics and Robotics Computing Group of the University of Tartu, which the author planned to use as a demonstration platform. ROS Melodic is paired with Ubuntu 18.04 and uses Python 2 by default. The author aimed to develop the code in Python 3, since Python 2 has reached its end of life. Therefore, Melodic was not the optimal ROS distribution and ideally, if another robot is going to be used in the future that has support for Noetic, the package should be used with ROS Noetic.

The D435i depth camera has a ready-made ROS wrapper, which is freely available in Intel's GitHub repository. The wrapper lets users interface the camera in ROS through broadcasting standardized ROS messages, for example the RGB and depth images. Although the wrapper is needed to generate the ROS topics to be used by the pipeline, the actual API (SDK) itself is also needed to access some of the library functions. This API, like the ROS wrapper, is freely available in Intel's GitHub repository.

As already mentioned, ROS uses a modular approach to robot software development. For this reason, the entire object detection pipeline was also built in a modular approach. An already existing RealSense node was used as well as a modified node of a 2D object detector. The author created two additional nodes as the core of the pipeline: 2D to 3D bounding box conversion node and a tracker node with included trajectory prediction. Furthermore, a debugging utility node for easy visualization was created as well as a separate program, that is not a ROS node, for annotating the data gathered from measurements. The whole solution will be presented and explained further after presenting its components separately and in detail.

3.2.3 Aligning Depth and Color

The D435i comprises both stereo infrared cameras and an RGB camera. As those different sensors cannot physically be in exactly the same position, the resulting images are naturally unaligned. Figure 3.2 illustrates the locations of the different sensors of the camera module, which include the right and left infrared imagers, an infrared projector and a color sensor. The X-Y center is located at the left IR imager and the Z center is located behind the camera's lens. For better understanding of the camera's different axes, they are shown in Figure 3.3, with X, Y and Z being designated with the colors red, green and blue, respectively. This difference in the positions of the different imagers is problematic, as the detector needs to know the depth value at each color pixel without any offset.

The RealSense ROS wrapper provides an easy way of addressing this problem. The camera's ROS node can be launched with an option of aligning the depth frame to the color frame. The node will then also publish additional topics for broadcasting the aligned image and the relevant camera information [28]. With this done, both the 2D object detector and the 3D bounding box finder can receive the same images with the depth image now matching the color image. This method is not without its problems, as will be discussed in the section about 3D sampling and bounding box prediction.

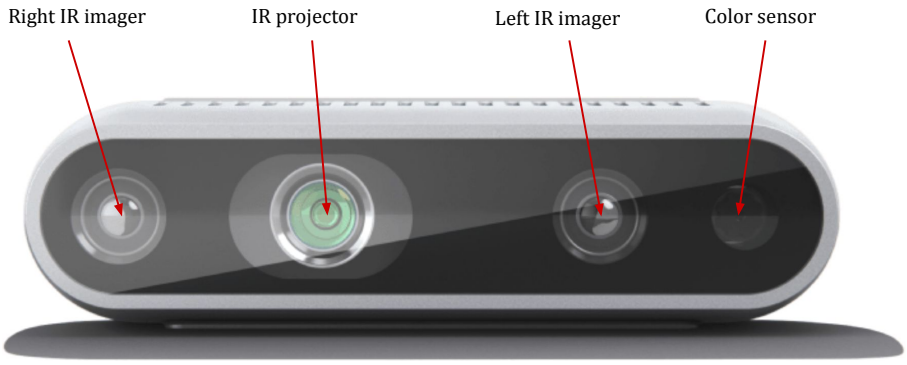


Figure 3.2: Components of the Intel RealSense D435i camera [23]

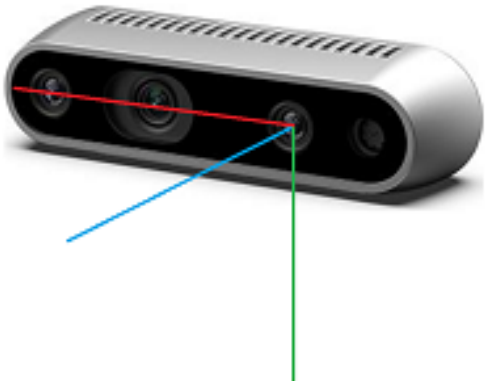


Figure 3.3: Axes of the Intel RealSense D435i camera [27]

3.3 Detection Using YOLOv5

Robust and reliable 2D object detection has been the subject of computer vision research for a long time. The premise is the analysis of sensor data, that is the image, to determine whether and where a given object exists. Although various objects can be detected, for this thesis the approaches are narrowed down to detecting humans, since discussing all the possible detectors is beyond the scope of this work.

A dichotomy exists between the type of algorithms used: artificial neural networks or traditional algorithms. Neural network approaches include Region Proposal networks (R-CNN and its successors), Single Shot (MultiBox) Detectors (SSDs), YOLO (You Only Look Once) and other combinations [5].

As is often the case, neural network based approaches require extensive custom training data in order to be useful in real applications. This can pose a problem of acquiring enough data for training. Since 2D and 3D detectors usually need different type of data for their respective operation, it also dictates which sensor have to be used in gathering of this data. For example, 3D detectors most often use LiDAR point-clouds as their input. Therefore training a 3D neural network based detector can require a large custom dataset, whereas training a 2D detector might only require custom 2D images of objects to be detected. The choice between the two can also take into consideration the sensor hardware used, for example on an existing robot which needs to be outfitted with a detector.

Although there is a variety of off-the-shelf 2D detectors available, YOLOv5 was chosen because

of its run-time performance and its reasonable detection accuracy. YOLOv5 was released by Jocher and others with the company Ultralytics in 2020 as a successor to Jocher's previous work on a PyTorch implementation of YOLOv3 [7].So far, no papers have been published of their work, although they do host an active GitHub repository and explain the implementation details in various issues [29, 7].

The YOLOv5 detector is an extension of the YOLOv3 PyTorch version. The latter is itself based on the widely known object detector YOLOv3 by Redmond and Farhadi [8]. The YOLOv5 network has a CSPNet (Cross Stage Partial Network) based backbone [30], a PANet (Path Aggregation Network) [31] neck and the head of YOLOv3 [29]. What differentiates the YOLO architecture from other DNN based object detectors is the single pass on the image. The bounding box anchors are generated prior to the detection itself, as opposed to R-CNN models, which makes the detection a lot quicker.

The detector is off-the-shelf trained and ready to be used but still there are some requirements to be satisfied before it can be actually used. It needs a GPU version of PyTorch for usable performance, the NVIDIA CUDA Toolkit and cuDNN (CUDA Deep Neural Network library) [7]. Also, a Python 3 environment with some additional numeric packages was needed.

After the installation, although the detector worked, it needed to be able to publish the found bounding boxes. For this, a part containing the publishing of ROS messages was added. The message Int32MultiArray of the standard messages package was used to not necessitate any additional third-party packages and to remain compatible with all ROS installations. Some other minor changes were made, for example adding the possibility to only output messages when at least one bounding box was found; pausing the detection for easier measurement taking.

Even though YOLOv5 was the detector of choice for this thesis, interfacing between the 2D detector 3D bounding box finder was intentionally left loosely coupled. This enables future research using different detectors for various needs, be it better detection accuracy of specific objects or even better real-time performance. It also enables the open-source robotics research community to swap out the components in the pipeline with considerable ease. Whatever detector is used, it only needs to publish to a topic the coordinates of the upper-left and bottom-right corners of any bounding boxes found.

3.4 3D Sampling and Bounding Box Prediction

Once the 2D bounding boxes have been received from the 2D detector, 3D bounding box prediction can begin. The core idea is simple yet effective: assuming a good enough 2D detection, naively sampling corresponding 3D coordinates of the detection will produce an accurate estimation of a 3D bounding box. The general idea and implementation will now be explained further, beginning with an overview of the camera's intrinsic matrix and the relationship of the image coordinates to those of the real-world, progressing to the implementation details

3.4.1 Depth, Camera Matrix and Deprojection

The basic principle of obtaining depth information from stereo vision follows the human binocular vision system. Depth is estimated by trying to find the difference in position of a fixed point, as viewed from two different sensors, called disparity [32]. The Intel RealSense camera does the

necessary processing on its integrated vision processor, lifting some computational load off the host computer. The calculation of disparity can be a difficult problem due to occlusions, and matching the two view-ports. To improve the performance of its cameras, Intel included an IR projector to boost the accuracy of the matching algorithms used to compute disparity [32]. Since depth is just directly related to depth, once the disparity of every point, and therefore also the depth, has been obtained, all coordinates of real-world objects can also be found, as is explained in the following paragraphs.

In a generalized way, the relation of 3D coordinates in the real-world to the 2D coordinates of points in an image taken by a camera can be described by the camera's projection matrix [33]. The projection matrix can be decomposed into the intrinsic and the extrinsic matrix. The former describes the geometry of the camera itself, while the latter describes the camera's location in the world. In the case of detecting the 3D bounding boxes based on 2D bounding boxes, the interest lies mainly in the intrinsic matrix, which can be written as:

$$K = \begin{bmatrix} f_x & 0 & 0 \\ s & f_y & 0 \\ c_x & c_y & 1 \end{bmatrix}$$

where f_x and f_y represent the focal length in pixels (the two do not have to be equal), c_x and c_y the principal point (center of projection) in pixels, and s is the skew coefficient [33]. In this model, the distortion is not accounted for and therefore the camera should output undistorted images.

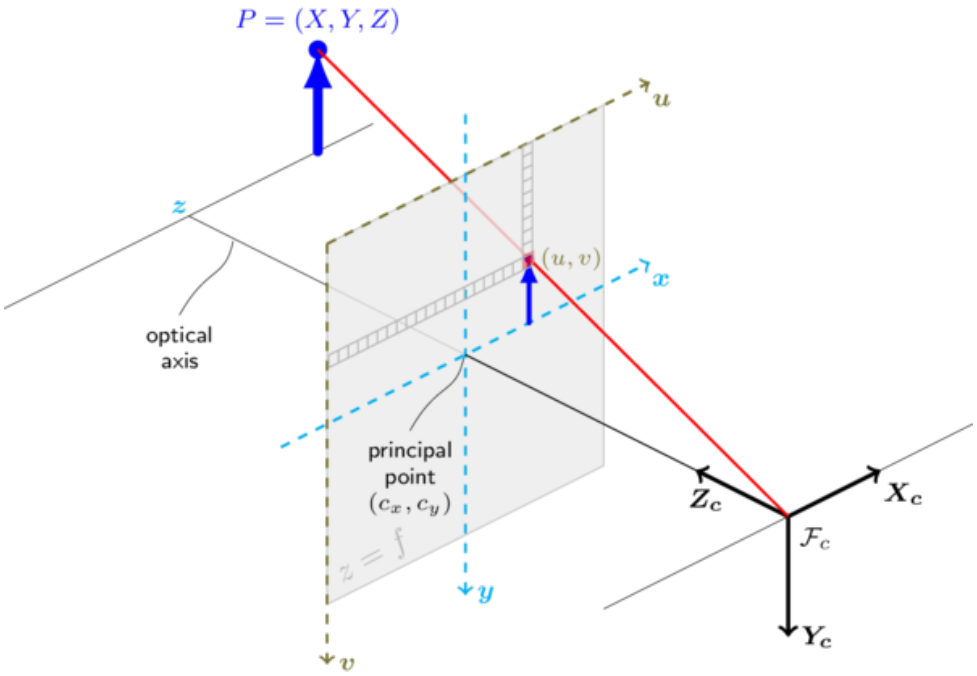


Figure 3.4: Model of the camera illustrating the concept of deprojection using similar triangles [34]

When the aforementioned parameters are known, only a single real-world value is needed to convert from the 2D plane to the 3D world. In the case of the Intel RealSense camera, these parameters are acquired during the program's execution. When a depth value corresponding to a

given pixel is known, the X and Y coordinates of the pixel can be converted into the real-world X and Y coordinates in an operation known as deprojection. In this case, the known real-world value is the depth in meters. The offset of the pixel from the principal point forms a triangle with the focal length in pixels. Another triangle is formed by the real-world distance of the same corresponding real-world point and the depth value. Therefore, the offset in both the X and Y directions of a real-world point can be calculated as follows:

$$x = \frac{(X - c_x)}{f_x} \cdot z$$

$$y = \frac{(Y - c_y)}{f_y} \cdot z$$

where x, y and z (depth) represent the real-world coordinates, X and Y the corresponding coordinates in pixels. This process is also illustrated in Figure 3.4, where u and v represent the pixel coordinates.

3.4.2 Implementation

The 3D bounding box prediction algorithm receives camera images and 2D bounding boxes via ROS topics. Although only the depth image is strictly required, it can also subscribe to the color image for easier visualization and debugging. Those images are converted from a ROS Image format to 2D arrays compatible with the Python library NumPy [35]. In a similar way, the received 2D bounding boxes are converted to tuples representing their corners' coordinates. Now, the algorithm can find the 3D bounding box based on the decoded inputs.

3D bounding box prediction comprises multiple steps. First, the depth image is sampled from the region the 2D bounding box occupies. For this, the coordinates of the samples are generated and then the actual sampling is carried out. The depth pixels are samples with fixed strides, and although it could be done dynamically, it does not offer considerable benefit, as the absolute number of samples is relatively low, so it is left as future work.

The process of converting the pixel coordinates into real-world coordinates is called deprojection and was discussed in the previous subsection. This is initially done in the program via calling a function from the RealSense Python library which outputted the X and Y coordinates with the corresponding pixel coordinates and the real-world Z coordinate as inputs. However, the performance of this version of the code was not satisfactory and improvements in the calculations were therefore made.

Both the proof of concept and final version of the programs written for this thesis have been written using the Python 3 programming language. Since the proof of concept work used sub-optimal Python operations, effort went into ways to improve the performance of the most critical sections of the program.

NumPy is an open-source Python package for scientific computing which leverages efficient array storage and operations [35]. NumPy allows for vectorization of operations performed on arrays. This quality of NumPy is at the core of the optimization performed for this thesis. In the rest of the section, the steps taken for optimization are explored and results are discussed.

The proof of concept program used nested native Python for-loops to sample the depth information and calculate the 3D bounding box coordinates. This was hypothesized to be one of the main

potential areas of performance improvement. To reduce the computational cost, vectorization with the use of NumPy was used. First, ranges of X and Y coordinates of pixels to be sampled were created. Then, the combination of those were used to vectorize the sampling of depth information. Furthermore, the implementation of a crucial function of the RealSense SDK was studied and its calculations were also vectorized using NumPy. The profiling of the critical part of the code showed a reduction of the computation time by about 2.5x.

After the coordinates have been found, a filtering is performed to eliminate invalid coordinates. Invalidity is defined in the following manner: if the device fails to determine the depth value of a pixel, the corresponding value in the depth map is set to zero [32]; therefore, the coordinates with the depth value of zero should be considered invalid samples and not be included in further calculation. The camera outputs zero depth when the depth was not valid in a given pixel location. If the depth image was perfect with every pixel being of valid depth, the process of finding the bounding box position would be much cheaper computationally, since multiple samples would not need to be taken, filtered, and then a final estimation made.

When the real-world coordinates have been filtered, a median in each axis is taken. The median essentially approximates a segmentation of the image depth-wise and as such, the final prediction is based on the object with the largest presence in the 2D bounding box. The idea of taking the median as an estimation of the true location of the object follows a simple assumption: if at least 50% of the area of the 2D bounding box is occupied by the detected object itself, the median in every axis has to correspond to some point on the object in each axis. This idea can be easily validated by the definition of the median, which is the value separating the higher and lower half of a data set. Since the objects detected in this thesis are humans, the assumption regularly holds, because humans do not usually occupy space in such a way that their projection to 2D occupies less than 50% of the area of their 2D bounding box. Figure 3.5 shows a detected human, with purple circles marking the 2D bounding boxes upper-left and lower-right corners. The blue dots illustrate the invalid coordinates which were filtered out and the red dots mark the coordinates on which the medians were calculated. As can be seen, the majority of sampled points are indeed situated on the human.

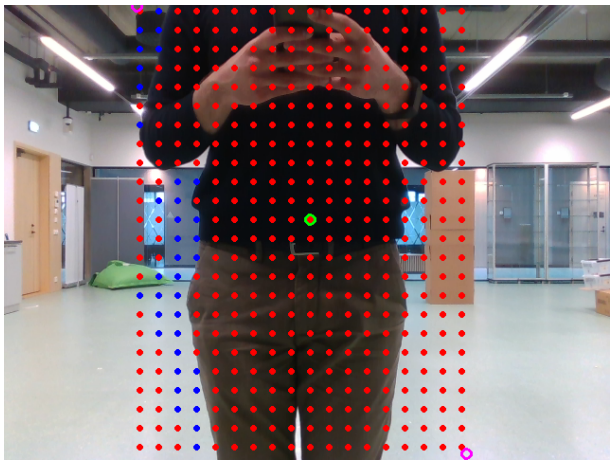


Figure 3.5: A detected human overlaid with sampled points

In addition to sampling the real-world coordinates, the size of the bounding box is also estimated. Using the obtained depth value, the X and Y values of the same depth are calculated for the 2D bounding box's upper-left and lower-right corners. Due to the view of the camera only

the bounding X and Y values can be sampled in a straightforward way with relative accuracy and therefore the Z dimension of the bounding box is estimated as being the same as the X dimension. This is, however, usually not a problem for the motion planning algorithms, as they often require smooth shapes with no local minima anyway and the polygonal bounding boxes would actually need to be reshaped [20]. Therefore, in later use, the 3D bounding boxes are technically cylinders.

The 3D bounding box detection node outputs, if specified, the unmodified color image, the modified color image for better visualization and debugging, the depth image and the 3D bounding boxes. The publishing of the multitude of different outputs can be turned on or off depending on the specific application. The 3D bounding boxes are broadcast as a custom ROS message BoundingBox3DArray, which, in turn, comprises the custom ROS messages of type BoundingBox3D. These custom messages were created as an alternative to the existing vision messages package, which lacked the support for 3D bounding box arrays for ROS Melodic. While, each published bounding box canonically consists of its position, orientation and size in all three dimensions, due to the Z and X dimension being identical and the bounding box actually being a cylinder, the orientation is always given as a normalized quaternion.

3.5 Visualization

Although the end goal of many robots is self-sufficient decision-making and navigation, the development process still requires the researcher to intervene. Often it is advantageous not to see the robot as a black box but in some way present its experience of the world in a human-understandable format. To do this, robotics researchers use custom or off-the-shelf visualization tools. One of the most popular of the latter is RViz, a 3D visualization tool for ROS [26]. However, to have a meaningful visual representation of the world, the perspective of the observer also matters.

One of the recurring problems in robotics is having different frames of reference for different parts of a robot. For example, a robot can describe the position and orientation, collectively referred to as the pose, of its arm in relation to its shoulder joint. When the robot moves itself in a global frame of reference, the arm's pose remains the same with respect to the robot's frame of reference, but not with respect to the global frame of reference. When now asked about the position and orientation of the arm in the global coordinate frame, the robot has to do some math. The more different coordinate frames there are, the more difficult it becomes keeping track of them all.

For visualization, this distinction between different frames of reference is also vital. In order for the researcher to debug the robot's behavior, a clear understanding must be present of what the world looks like to the robot and how the robot is navigating in relation to the world.

3.5.1 Overview of Coordinate Frames in ROS

To combat this issue and aid robot software development, a ROS package called tf2 was created. The tf2 package lets users broadcast and listen to coordinate frames. The existing coordinate frames can be displayed as a graph and tf2 allows users to convert a pose from one coordinate frame to another. This can be as simple as listening to broadcasted coordinate frames, choosing the two coordinate frames to transform between and applying the transform on a pose [36].

Coordinate frames in tf2 are the vertices of a coordinate frame graph. Each coordinate frame, called the child frame, except the root, has a parent frame. The child frames are related to their parent frames by some given translation and rotation. This gives rise to the ability to transform any coordinate frame to any other coordinate frame if those coordinates frames in the coordinate frame graph are connected [36].

As already mentioned, robots usually have more than one coordinate frame. In the case of this thesis, one of the coordinate frames is that of the camera. As was shown in Figure 3.3, the Z axis corresponds to the depth, the X axis the horizontal and Y the vertical direction. In the case of ROS, the coordinate frame of the robot is different. Figure 3.6 shows the ROS coordinate frame of a robot, where the X axis is straight ahead for the robot, Y points to the left and Z upwards. The axes are color-coded as red, green and blue, respectively. The camera's axes are also overlaid as thinner lines. The point of origin of the robot's coordinate frame is often called the base link and it typically corresponds to the rotational center of the robot [37].

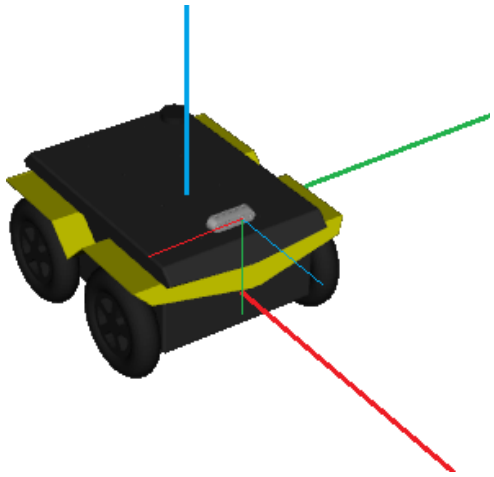


Figure 3.6: Axes of Clearpath Jackal and a mounted Intel RealSense D435i camera [38]

In addition to the different configuration of axes, the camera is also probably not positioned directly at the coordinates of the base link, rather mounted somewhere else on the robot. In the case of Jackal, for example, the X position is 17 cm forward of the base link and the Z position is also offset by 20 cm. In this case, the coordinate frame of the camera is defined in relation to the base link coordinate frame.

The discussion has so far focused on the relation of the camera's and robot's coordinate frames. But robots themselves must also have parent coordinate frames if they are to move around in the world. The parent of base link is called odom, which essentially means the odometry of the robot. Although accurate in the short-term, odometry can drift and therefore something else should be used as a more global frame of reference. Therefore, map is often used as the parent of odom. Map represents the world as a fixed frame. There is also a coordinate frame called earth, but for the purposes of this thesis, earth is not required [39].

3.5.2 Simple Example

In the previous subsection an overview was given of the coordinate frames in ROS with some examples of the setup used in this thesis. The use of tf2 and different coordinate frames solve

the problem of converting the 3D coordinates of the detections from the camera's coordinate frame to the robot's coordinate frame, or even better, the world's coordinate frame. As robots navigating in dynamic environments means that they move around, just knowing the location of an object in the robot's frame of reference is not sufficient for understanding of the dynamics of the detected objects. For example, when the robot itself is moving and the an object is stationary, then in the robot's frame of reference the object appears to move. This is however not desirable for neither visualization nor tracking, as is now further discussed.

In Figure 3.7, a screenshot of a simulation is shown. The robot Jackal is shown to detect a single object with its Intel RealSense camera. The detected human is about a two meters away from the robot and as can be seen, a blue cylinder is created to visualize the detection (bounding box). The given example illustrates some major limitations of using just an object detector for robot navigation: in the presence of multiple objects, the robot cannot semantically tell the objects apart and cannot therefore identify their respective movements. Furthermore, when the detection is in the coordinate frame of the camera, the robot's movements cannot be differentiated from the movements of the detected objects. The latter problem, as discussed, can be solved by transforming between different coordinate frames. The former is solved by object tracking.

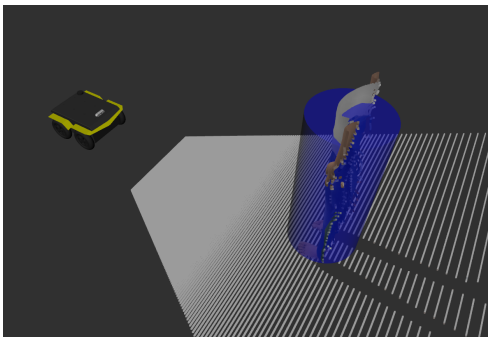


Figure 3.7: Visualization of Jackal detecting a human in a simulated environment

3.6 Object Tracking and Trajectory Prediction

Object tracking is the assignment of unique identifiers to detected objects and keeping track of their movements. Object tracking is needed to have a temporal understanding of the objects populating the environment. There have been various proposed tracking methods, ranging from complex neural network based approaches [40] to simple traditional algorithms, such as the one presented in this thesis. As with object detection, there is a trade-off between efficiency and accuracy. In order for the robot to make decisions real-time, the tracker needs to work at a relatively high frequency. Although the approaches can be complex, the running times can be in the order of magnitude faster than those of the detectors, so the performance of the tracker may not be as critical compared to that of the detector. The focus of this thesis was on object detection, however, a working object tracker with basic trajectory prediction was also developed to demonstrate a complete and immediately usable solution.

The tracking algorithm used in this thesis is simple, yet effective. The core idea is matching objects based on their positions between consecutive frames. The X and Y positions of objects are tracked in the world coordinate frame (bird's-eye view) and therefore the solution is much more robust than a typical tracking algorithm operating on a side-view 2D projection. Although

the idea was of the author's own making, a well-optimized version of the same basic algorithm was found and parts of it used [41]. The positions of objects in two consecutive frames are compared, pairs with smallest distances between existing and a new detections are found, and, starting from the pair with the smallest distance, each existing detection is updated with the corresponding new. The algorithm does not, however, go through all possible combinations, that is find a globally optimal solution to the problem, which generally is referred to as the linear assignment problem [42]. It instead makes some assumptions to ensure its overall effectiveness:

- The walking speed of humans does not normally exceed 2.5, rounded to 3 m/s [43].
- The object detection pipeline runs at 30 frames per second.
- Minimal dimensions of a human cannot be under 0.1 m in any direction.

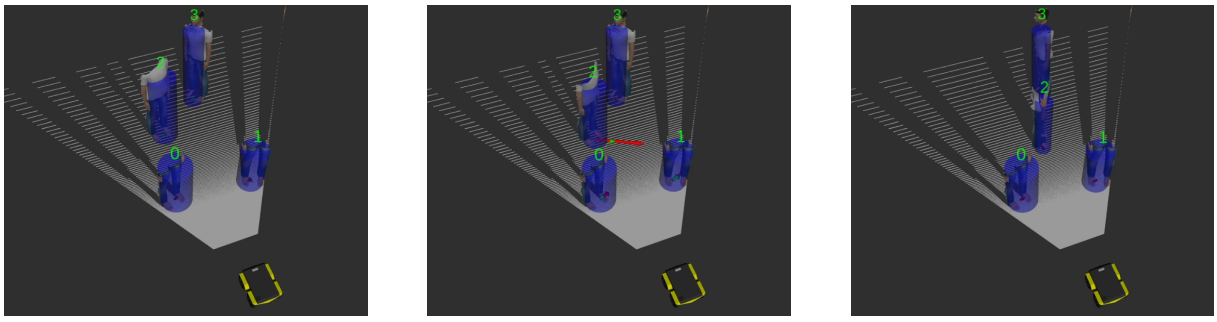
Therefore, it can be deduced, that a detected human's position cannot change more than 0.1 m between two consecutive frames. If this assumption holds, then two humans are not able to swap their positions during a single detection period. As such, the closest match for each detected object cannot be closer to any other object, removing the need for a global optimum, allowing the use of a more efficient approach. Also, when a match is not found for an existing object, it will be deleted unless a match is later found in a specified period. Any new detections which do not get matched to an existing one are considered as new objects.

For efficient motion planning, in addition to detection and tracking, trajectory prediction is also required. Trajectory prediction algorithms try to estimate the positions the objects will occupy in some specified amount of time. Knowing which positions obstacles can occupy in the future lets motion planning algorithms choose the path with the lowest control cost to the desired destination. Although sophisticated neural network based approaches for human trajectory prediction exist [44], for this thesis, a crude trajectory prediction algorithm was integrated into the solution to illustrate the potential of trajectory prediction.

The trajectory prediction algorithm performs low-pass filtering on the velocities of the objects, which are computed every time the tracking algorithm receives new predictions. The trajectory prediction algorithm uses exponential averaging to filter out fluctuations of the estimated positions of objects caused by the neural network based 2D object detector. Then, a linear prediction is made based on the current position of an object and its filtered velocity. The sensitivity of the filter can be tweaked using the parameter α to make the prediction either more (α increased) or less (α decreased) reactive to new data. The exponential moving average is an infinite impulse response filter and was chosen because of its inexpensiveness regarding computation time and the intuitiveness of its single calibration parameter.

4 Simulation Results

The complete detection pipeline along with object tracking and trajectory prediction was tested both in real-life using the Intel RealSense D435i camera and in Gazebo, a robot simulator [25]. Multiple human detection and tracking is shown only in the Gazebo simulation.



(a) Starting position (b) Object 2 moves (c) Objects 2 and 3 near

(d) Object 3 passing 2 (e) Object 3 passed 2 (f) Object 0 passed in front of 3

Figure 4.1: Scenes demonstrating the 3D bounding cylinders, tracking and trajectory prediction

Figure 4.1, shows six simulation experiments demonstrating the capabilities of the developed detection pipeline. The scene is set up with four human objects and the Jackal robot with a simulated Intel RealSense camera. The pipeline was run on the simulated camera feed and the produced outputs were visualized using RViz. The images are screenshots of the output of RViz. Figure 4.1a shows the starting positions of the objects. After the objects were set up, they were moved gradually to new positions. Each object is bounded with a blue cylinder which is the visual representation of the output of the 3D object detection pipeline. Above the cylinders are green number representing the unique IDs obtained from the tracking algorithm.

In Figure 4.1b, object two can be seen moving. Movement is indicated by a red arrow, corresponding to the instantaneous velocity of an object. A partially visible green arrow can also be seen, which is the visualization of the output of the trajectory prediction algorithm, marking the

spot the object should occupy in a given number of frames or seconds (in this case two seconds). Figure 4.1c shows the objects two and three approaching each other. Figure 4.1d again shows a green arrow predicting the objects position in two seconds. The arrow is relatively short because small steps were taken moving the objects. By Figure 4.1e, objects two and three have passed each other from the 2D perspective of the camera. Situations like this are challenging from tracking perspective. However, as can be seen, the developed tracking algorithm is robust enough to still identify the objects with their correct IDs.

Finally, Figure 4.1f shows a situation where object zero is passing in front of objects two and three. As objects move close to or pass in front of each other, the 2D bounding boxes from the 2D object detector will also get somewhat changed in dimensions. This causes the 3D detection algorithms to slightly misestimate the positions and dimensions of the detected objects, resulting in considerable spikes in their velocities, as can be seen from long red arrows. However, the trajectory prediction algorithms easily suppresses these sudden changes.

In addition to showing the results of a demonstration, performance in terms of frames per second

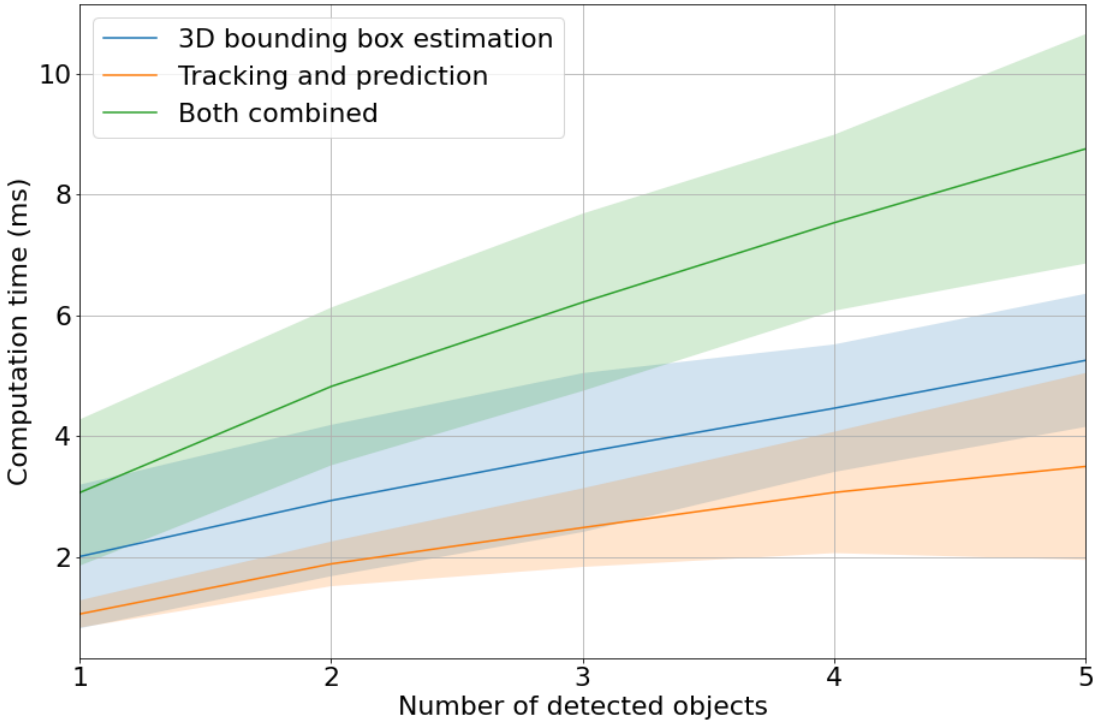


Figure 4.2: Computation times with regard to the number of detected objects

is analysed. In Figure 4.2 the mean computation time of the developed programs is visualized. The highlighted areas around each curve represent one standard deviation above and below the corresponding mean. The timings of the 3D bounding box prediction, and tracking and trajectory prediction are given, as well both of them combined. As can be seen, the combined average computation time takes under 10 ms even in the case of five detected objects. Analysing the results, the combined performance of both programs running simultaneously can be in the range of 100 – 300 frames per second or possibly even more. Including YOLOv5, the average total computation time of the pipeline was under 40 ms. Considering that YOLOv5 ran on an average frequency of about 30 frames per second, it can be safely assumed that finding the 3D bounding box based on the 2D bounding box while also tracking and making predictions about the positions of the tracked objects will not not be a bottleneck. This also shows potential for

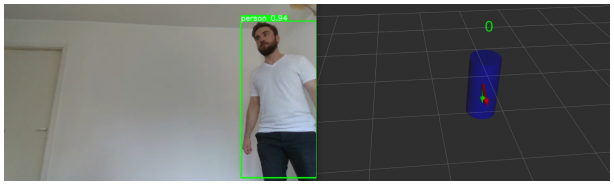
future work exploring different 2D detectors or hardware acceleration for integration into the pipeline for further reducing the total computation time.

4.1 Real-World Experiments

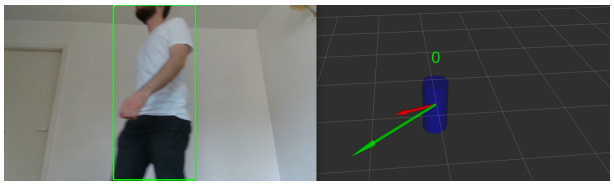
To complement the simulation results and to demonstrate the working of the developed pipeline with real sensor inputs, a small presentation is given using the Intel RealSense D435i camera and a human moving in the scene.

In Figure 4.3, outputs of both the 2D object detector and the developed detection pipeline along with tracking and trajectory prediction can be seen, visualized using RViz. Figures 4.3a, 4.3b and 4.3c show the outputs of the first recorded scene and Figures 4.3d, 4.3e and 4.3f show the outputs of a second scene. The left side of each image shows the 2D bounding boxes generated by YOLOv5. On the right side of each image an RViz visualization of predicted 3D bounding boxes (visualized as cylinders), unique IDs, as well as velocities and predicted positions can be seen. As with the simulated data, red arrows show the instantaneous velocity of an object, while green arrows show the estimated position of an object in some given amount of frames (time).

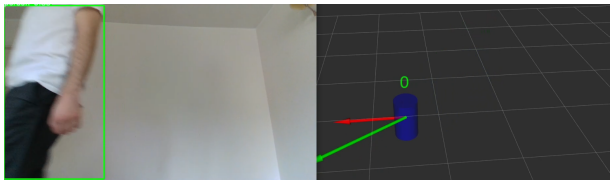
As can be seen from Figure 4.3, the instantaneous velocity of an object can seem chaotic. This is due to the noise from the neural network based 2D detector. To counter this issue, the trajectory prediction algorithm filters out these fluctuations to produce more accurate estimations of future positions. Also, as shown, the visualized cylinders shrink and expand. This behavior can also be attributed to the 2D detector: comparing Figures 4.3d and 4.3e, it can be seen that in the former, the 2D bounding boxes is fitted tightly around the object, while in the latter an extending arm causes the bounding box to increase in its size horizontally. For robot navigation, this behavior can be useful for avoiding obstacles with changing dimensions.



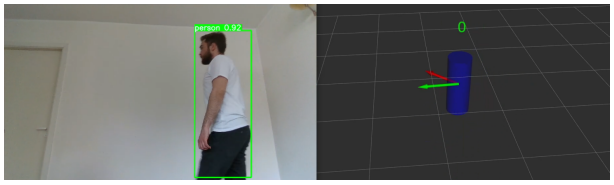
(a) Scene 1 starting position, object moving slightly



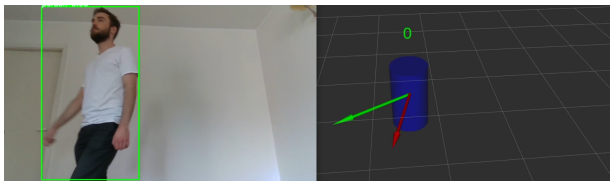
(b) Object moving



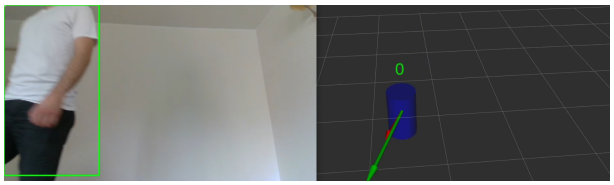
(c) Object going out of the frame



(d) Scene 2 starting position, object already moving



(e) Object moving while changing its direction



(f) Object going out of the frame

Figure 4.3: Two scenes demonstrating the 3D bounding cylinders, tracking and trajectory prediction with real-world data. On the left side, output of the 2D object detector can be seen. On the right, an RViz visualization

5 Noise Analysis

The objective of this section is to characterize the noise associated with the proposed 3D bounding box detection pipeline. The source of the noise stems from the error in the RGB-D values which then gets mapped to the 3D bounding boxes. This chapter is specifically focused on empirically deriving the noise distribution of the detection pipeline. This, in turn, is crucial for developing probabilistic motion planning algorithms such as [45] that explicitly use the noise distribution information to characterize the probability of collision avoidance associated with a computed trajectory.

For this thesis, a measurement process was planned and carried out. Based on the measurement data, a thorough noise analysis was also conducted. Both the measurement process and the interpretation of the data are explored in the following sections.

5.1 Measurement Process

The measurement process was designed to reasonably characterize the accuracy of the detection pipeline while still being feasible to conduct. The high-level overview of the measurement process is illustrated in the figure 5.1. The gray oblong object marks the camera, while the orange circles represent the positions a human will occupy during a specific measurement. Also, a grid can be seen which makes it easier to see the coordinate of the human's position in relation to the camera.

In total, two separate measurement takings took place, since during the first one there was a miscalibration of the detection pipeline and the results were not reliable. The second time the whole detection pipeline worked correctly and data was successfully gathered for further analysis. The measurement itself was conducted by the author and a detailed description is given next. First, the Intel RealSense camera was fixed to a table and connected to a laptop for interfacing. The camera's tripod was aligned in such a way that the optical center line was aligned with a straight line in the flooring of the room. Next, the positions the human would occupy were measured and marked with masking tape. After the marking, the measurement process could begin.

While a person stood in the designated spot, snapshots of both the color and depth images with detection results were taken. For each marked spot, at least three measurements were taken, with six being the most frequent amount of repetitions. In total, 184 discrete measurements were taken during the main measurement process. Before this, another experiment was carried out which unfortunately had unreliable estimations in one axis due to incorrect calibration of the detection pipeline. However, 161 measurements in one axis were still valid and were used to augment the newer data. As for the measurement accuracy, the ground truth marking were set using a tape measure, therefore the error could be expected to be in the range of a few millimeters.

Also, the person standing on the desired positions tried to align his body to be at the desired distance from the camera, while being centered in the other direction. Naturally, the described process has some inherent inaccuracies. Nevertheless, for characterizing the detection pipeline, this level of accuracy was deemed adequate.

After the measurement process took place, the gathered data needed to be annotated. For assisting the annotation, a simple script was created which shows the user the pictures and the data and asks to enter the ground truth. The program also automatically calculated the errors. All the measurements were processed in the described manner and all the data was exported to a spreadsheet file. The gathered data is present in a GitHub repository and available for future work [22].

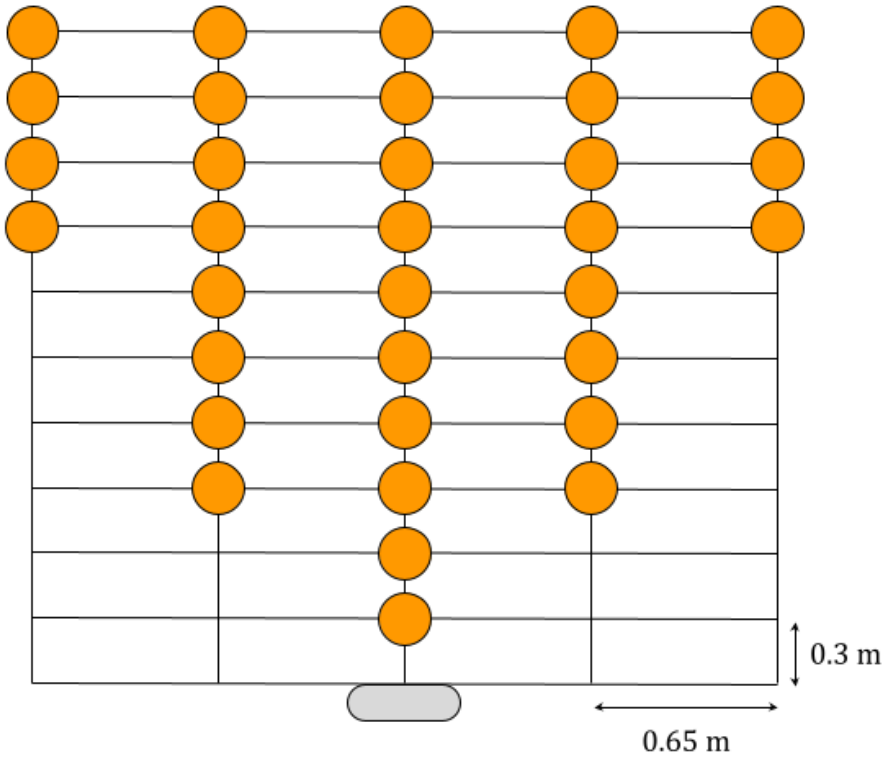


Figure 5.1: Positions of the human in relation to the camera from a bird's-eye view (not to scale)

5.2 Results

For this thesis, only the X and Z axes of the camera were considered, since those correspond to the movement of objects on the ground plane. The accuracy in the Y direction is more heavily limited by the vertical field of view of the camera and objects such as humans are often vertically partially out of the frame. This leads to lesser accuracy in the Y direction and therefore it was not considered, although it can be analysed as future work.

Stereo cameras have an inherent systematic error in the depth direction. This is known as the distance inhomogeneity [16]. The equation for the RMS depth error is given as:

$$e = \frac{d^2 \cdot s}{f \cdot b}$$

$$f = \frac{1}{2} \frac{X_{res}}{tan(\frac{FOV_H}{2})}$$

where d is the distance from the camera in mm, s the empirically determined sub-pixel accuracy, f the focal length in pixels, b the baseline in mm, X_{res} the horizontal resolution in pixels and FOV_H the horizontal field of view. [46]

As the depth image is aligned to the color image, the color sensor's horizontal field of view has to be used. The camera's datasheet specifies it as 69° and the baseline is specified as 50 mm. As mentioned, the sub-pixel value is determined empirically, because it is dependent on the calibration of the specific camera. Intel recommends to use the value 0.08 for the D435 series camera. [47, 46] In the case of this thesis, the horizontal resolution is set as 640 pixels.

Figure 5.2 shows both the theoretical and empirical depth RMS error as well as a curve fitted to the empirical data for a number of different distances. The fitted curve begins at 1.5 m distance, as the shorter distances seem not to relate to the distance quadratically. The formula for the curve is provided in the figure label. As can be seen, the observed RMS value is higher than the theoretical regardless of the underlying distance. This can be explained by the methodology of the measurements in both cases: the theoretical RMS assumes a flat plane, while the empirical data gathered for this thesis measured the distance of a real human [48].

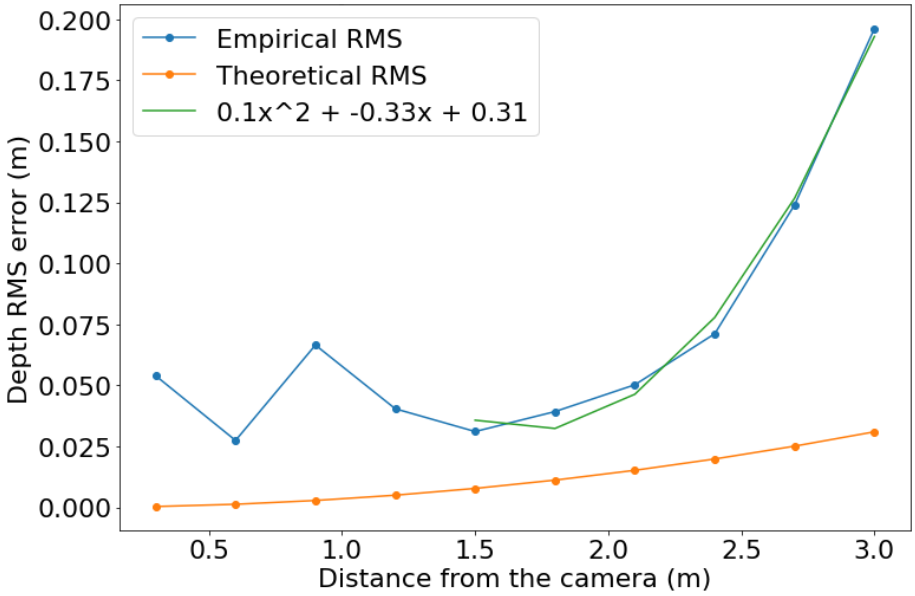


Figure 5.2: Depth distortion vs distance, theoretical vs empirical

It is reasonable to assume that the detection noise caused by the neural network based 2D detector also increases the error. Nevertheless, the results are mostly aligned with the theory of stereo cameras. It is important to note that the referred theoretical RMS is the theoretical minimum error of stereo depth cameras and therefore in practice the error is expected to be greater. [46]

So far, the theoretical limitation of the RMS error has been analysed. However, characterizing the distribution parameters such as mean and variance is also important for robotic applications. In the following, this analysis is presented and also the general trend observed in these parameters is discussed.

Figure 5.3 shows the Z (depth) error observed at different Z measurements. The mean of the error distribution is found to be quadratically dependent on the distance and can be modeled as the following function $f(z) = -0.04z^2 + 0.06z + 0.008$, where z is the distance from the camera in meters. The quadratic nature of the mean error with regard to distance is also supported by the camera's developers [46]. The mean function is, however, not enough to fully characterize the noise. To create a probability distribution of the error, the variance is also needed.

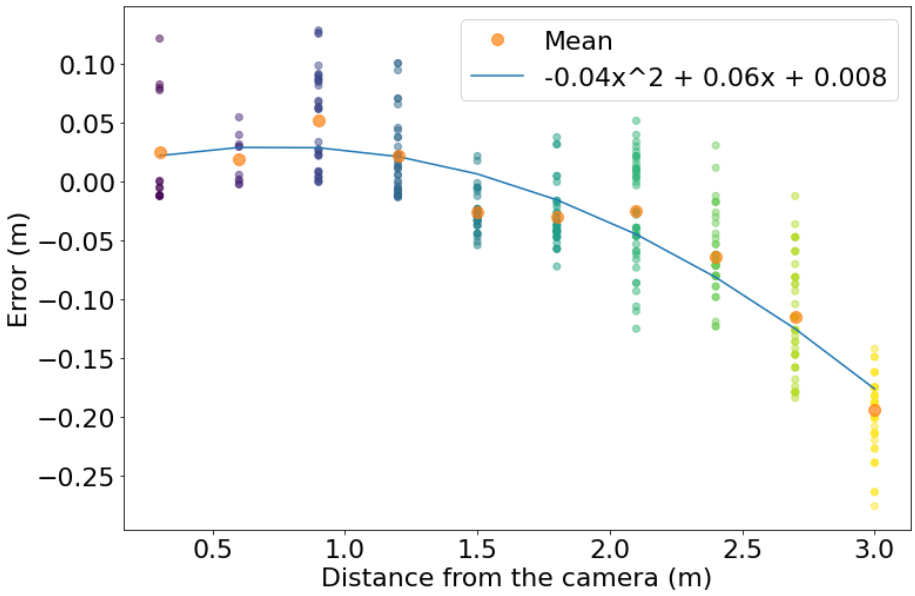


Figure 5.3: Z error vs Z distance with error bars of one standard deviation

Figure 5.4 shows the how the variance of the error is related to the distance from the camera. As can be seen, no qualitative trend exists between the variance of the noise and the distance. Therefore, different values can be used to obtain various levels of confidence when estimating an object's position. Based on the data, the maximum variance of the error was found to be $\sigma^2 = 0.00225\ m^2$ and the standard deviation of $\sigma = 0.0474\ m$ or about 5 cm. Additionally, it should be noted that this variance is calculated based on 345 measurements arranged into 10 clusters.

Figure 5.5 shows the X (horizontal) error observed at different X measurements. As shown, a linear fit can be observed for the mean error with the estimated error function being $f(x) = -0.106x - 0.006$, where x is the horizontal distance from the camera. Again, the error bars of one standard deviation are also shown.

Unlike Z variance, X variance seems to be related to the distance from the optical center, as is shown in Figure 5.6. The author offers no conclusive reason for this behavior but this systematic error could actually be non-systematic and caused in the unequal distribution of the different distances in the data set, which were 139 in total. Furthermore, the X variance can be increased by edge-of-the-frame detections, where part of the human is out of the frame and therefore the measured value is offset from the ground truth, although such samples were generally avoided. Also, the noise could be influenced by the 2D neural network based object detector and also some calibration issue with the camera. Nevertheless, the maximum variance and standard deviation were calculated for use in motion planning: $\sigma^2 = 0.00032\ m^2$, $\sigma = 0.018\ m$.

Ahn et al. have also performed noise modeling of the Intel RealSense D435 camera. They

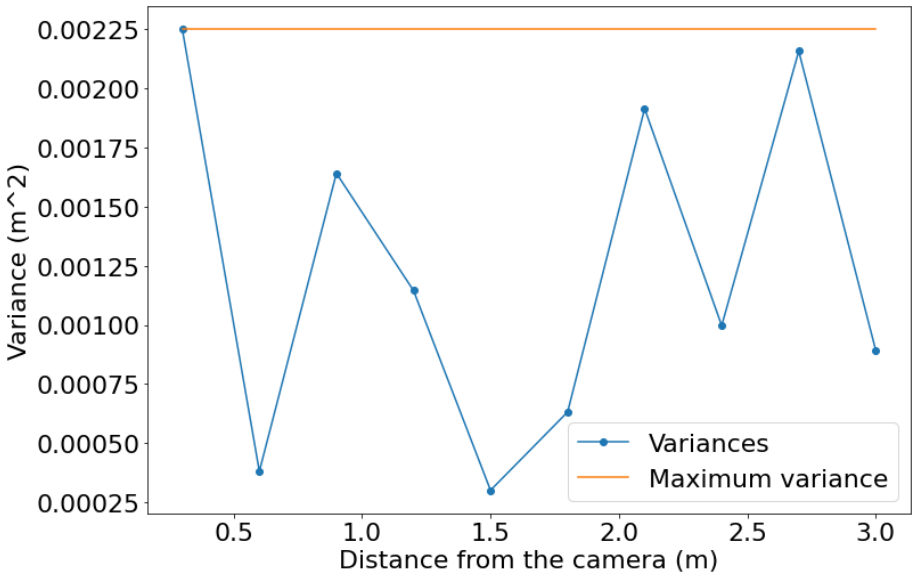


Figure 5.4: Variances of different clusters of measurements

claim the error distribution can be shown to be Gaussian and therefore can be used in versatile appliction in mapping [16]. However, analysing the measurement data collected for this thesis, the same conclusion cannot be easily drawn. To show this, normalized histograms of the data were overlaid with their respective best-fit Gaussian distribution's probability density functions. A possible reason for non-Gaussian error distribution in the present case could stem from the fact that the 3D bounding box detection pipeline consists of several non-linear transformations of the raw RGB-D values. This includes the operations that take place within the YOLOv5 detection pipeline and the filtering process used to correlate the 2D bounding boxes of YOLOv5 with the corresponding depth values to estimate 3D bounding boxes.

Unlike Min Sung Ahn et al., this thesis did not find any conclusive evidence for the error distribution to be Gaussian in nature [16]. Figure 5.7 shows the error distribution in the case of different measurements in the Z direction. As shown, most of the data on histograms cannot be adequately modeled as Gaussian distributions, although some examples resemble it. Considering the amount of data collected, these results are certainly not enough to rule out any possibility of the error distribution to be Gaussian, but in this case, the noise certainly cannot exactly be modeled as such and some other assumptions will have to be made.

Similar histograms were created to show the distribution of the error in the X direction. Figure 5.8 shows these histograms. As can be seen, again the distributions, although bearing a resemblance, cannot be said to be exactly Gaussian. Further data gathering and analysis is needed to draw further conclusions. Nevertheless, the previously shown maximum standard deviations can be conservatively used in both the X and Z error estimation for motion planning algorithms.

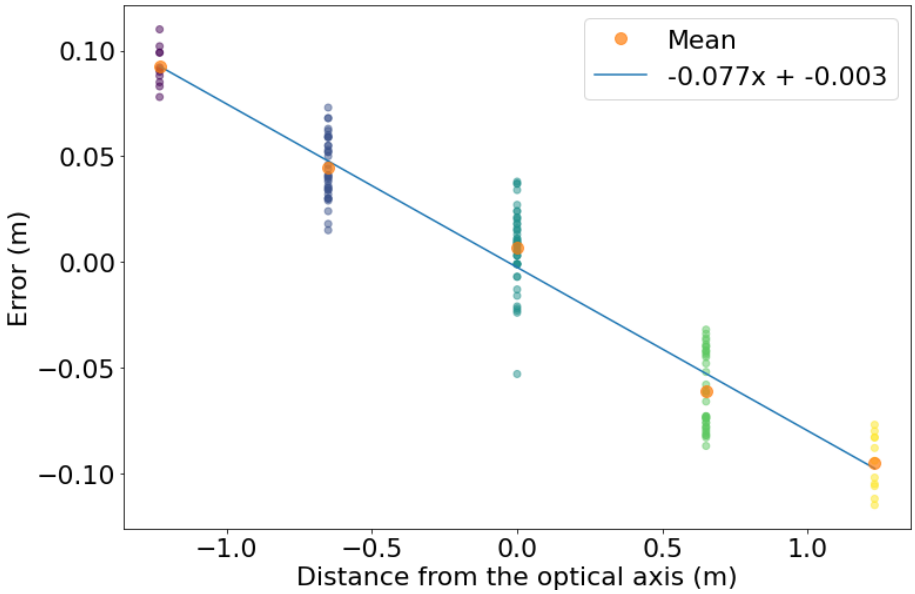


Figure 5.5: X error vs X distance with error bars of one standard deviation

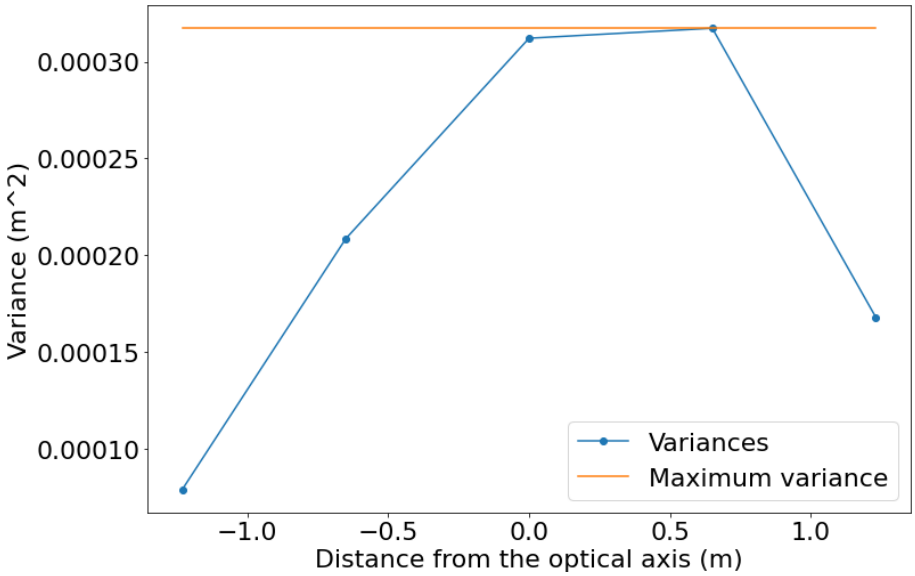


Figure 5.6: Variances of different clusters of measurements

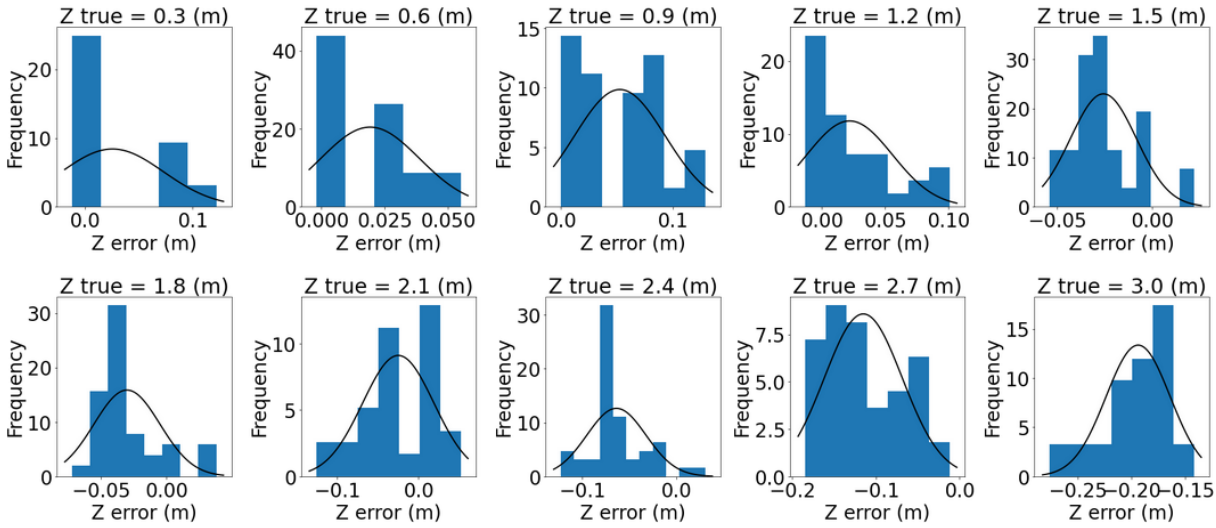


Figure 5.7: Distributions of error for different values of Z distance

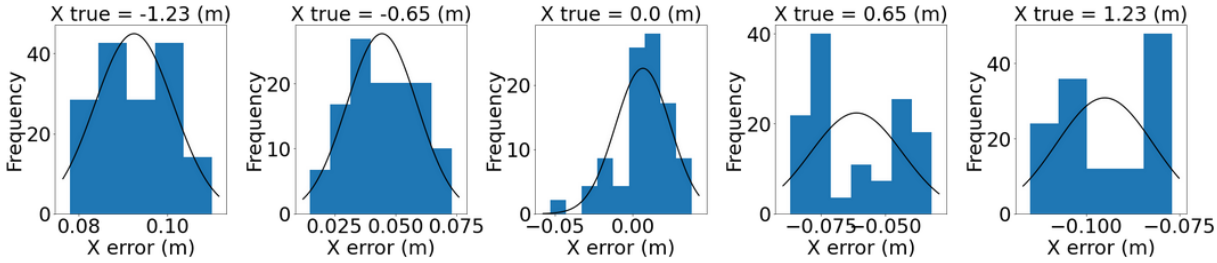


Figure 5.8: Distributions of error for different values of X distance

6 Conclusions and Future Work

This thesis developed a complete software pipeline for detection of humans in a scene based on the input of a depth camera followed by estimating their 3D positions and creating bounding boxes (or cylinders) around them. The efficacy of the developed software pipeline was validated in both simulation and real-world experiments. Furthermore, it runs real-time on an NVIDIA RTX 2070 GPU-enabled laptop.

The immediate future endeavour is geared toward integrating the detection pipeline with a motion planning algorithm and demonstrating navigation of a mobile robot in crowded environments. The work in this thesis will be also be applied to reactive navigation of quadrotor drones. In this context, the detection pipeline will be extended to incorporate not only humans but also other obstacles, such as walls, trees (in outdoor forests) etc. Other possibilities for future work include using different 2D detectors with either greater accuracy or better real-time performance, and dynamic sampling of real-world coordinates in the 3D bounding box prediction algorithm.

Acknowledgements

I want to thank

my supervisor Arun Kumar Singh for introducing me to the fascinating world of research as well as guiding my work throughout this whole process,

Jatan and Houman for their help, guidance and encouragement,

and finally the University of Tartu for probably one of the best Bachelor's programmes in Estonia.

Bibliography

[1] K Mohanaprakash et al. "Computerized Robot for Ground Navigation in Hospital Buildings". en. In: *International Journal of Engineering Research and General Science* 4.1 (2016).

[2] Ali Gürcan Özkil. "Service Robots For Hospitals: Key Technical Issues". en. PhD thesis. Technical University of Denmark, Apr. 2011.

[3] Takayuki Kanda et al. "A Communication Robot in a Shopping Mall". In: *IEEE Transactions on Robotics* 26 (Oct. 2010), pp. 897–913. DOI: 10.1109/TRO.2010.2062550.

[4] Lorenzo Nardi and Cyrill Stachniss. "Long-Term Robot Navigation in Indoor Environments Estimating Patterns in Traversability Changes". en. In: *arXiv:1909.12733 [cs]* (Sept. 2019). URL: http://arxiv.org/abs/1909.12733 (visited on 05/19/2021).

[5] Zhong-Qiu Zhao et al. "Object Detection with Deep Learning: A Review". en. In: *arXiv:1807.05511 [cs]* (Apr. 2019). URL: http://arxiv.org/abs/1807.05511 (visited on 05/19/2021).

[6] *Explained: Neural networks*. en. URL: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (visited on 05/19/2021).

[7] *ultralytics/yolov5*. May 2021. URL: https://github.com/ultralytics/yolov5 (visited on 05/17/2021).

[8] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". en. In: *arXiv:1804.02767 [cs]* (Apr. 2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767 (visited on 05/18/2021).

[9] Charles R. Qi et al. "Frustum PointNets for 3D Object Detection from RGB-D Data". en. In: *arXiv:1711.08488 [cs]* (Apr. 2018). URL: http://arxiv.org/abs/1711.08488 (visited on 05/17/2021).

[10] Lin Yan et al. "RTL3D: real-time LIDAR-based 3D object detection with sparse CNN". en. In: *IET Computer Vision* 14.5 (2020), pp. 224–232. ISSN: 1751-9640. DOI: https://doi.org/10.1049/iet-cvi.2019.0508. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cvi.2019.0508 (visited on 05/16/2021).

[11] Athanasios Voulodimos et al. "Deep Learning for Computer Vision: A Brief Review". en. In: *Computational Intelligence and Neuroscience* 2018 (2018), pp. 1–13. ISSN: 1687-5265, 1687-5273. DOI: 10.1155/2018/7068349. URL: https://www.hindawi.com/journals/cin/2018/7068349/ (visited on 05/19/2021).

[12] Marcel Cata Villa. "3D Bounding Box Detection from Monocular Images". MA thesis. Stockholm: KTH School of Electrical Engineering and Computer Science, July 2019. URL: http://www.diva-portal.org/smash/get/diva2:1358670/FULLTEXT01.pdf (visited on 05/16/2021).

[13] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *arXiv:1506.01497 [cs]* (Jan. 2016). URL: http://arxiv.org/abs/1506.01497 (visited on 05/16/2021).

[14] Xinzhu Ma et al. “Accurate Monocular Object Detection via Color-Embedded 3D Reconstruction for Autonomous Driving”. en. In: *arXiv:1903.11444 [cs]* (Mar. 2021). URL: http://arxiv.org/abs/1903.11444 (visited on 05/16/2021).

[15] Waleed Ali et al. “YOLO3D: End-to-end real-time 3D Oriented Object Bounding Box Detection from LiDAR Point Cloud”. en. In: *arXiv:1808.02350 [cs, eess]* (Aug. 2018). URL: http://arxiv.org/abs/1808.02350 (visited on 05/17/2021).

[16] Min Sung Ahn et al. “Analysis and Noise Modeling of the Intel RealSense D435 for Mobile Robots”. In: *2019 16th International Conference on Ubiquitous Robots (UR)*. ISSN: 2325-033X. June 2019, pp. 707–711. DOI: 10.1109/URAI.2019.8768489.

[17] *OpenCV: Cascade Classifier*. URL: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html (visited on 05/20/2021).

[18] *OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform)*. URL: https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html (visited on 05/20/2021).

[19] Carlo Tomasi. “Histograms of Oriented Gradients”. en. In: (). URL: https://courses.cs.duke.edu//fall15/compsci527/notes/hog.pdf (visited on 05/20/2021).

[20] Jiahao Lin, Hai Zhu, and Javier Alonso-Mora. “Robust Vision-based Obstacle Avoidance for Micro Aerial Vehicles in Dynamic Environments”. en. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Paris, France: IEEE, May 2020, pp. 2682–2688. ISBN: 978-1-72817-395-5. DOI: 10.1109/ICRA40945.2020.9197481. URL: https://ieeexplore.ieee.org/document/9197481/ (visited on 05/16/2021).

[21] Adarsh Jagan Sathyamoorthy et al. “COVID-robot: Monitoring social distancing constraints in crowded scenarios”. In: *arXiv:2008.06585* (2020).

[22] Jyrijoul. *Jyrijoul/ros_3d_bb*. URL: https://github.com/Jyrijoul/ros_3d_bb (visited on 05/20/2021).

[23] *Depth Camera D435i*. en-US. URL: https://www.intelrealsense.com/depth-camera-d435i/ (visited on 04/29/2021).

[24] Morgan Quigley, Brian Gerkey, and William D. Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. 1st. O’Reilly Media, Inc., 2015. ISBN: 1-4493-2389-8.

[25] *Gazebo*. URL: http://gazebosim.org/ (visited on 05/20/2021).

[26] *rviz - ROS Wiki*. URL: http://wiki.ros.org/rviz/ (visited on 05/20/2021).

[27] *Calibration Tools User Guide for Intel® RealSense™ D400 Series*. en. URL: https://dev.intelrealsense.com/docs/intel-realsensetm-d400-series-calibration-tools-user-guide (visited on 05/20/2021).

[28] *IntelRealSense/realsense-ros*. May 2021. URL: https://github.com/IntelRealSense/realsense-ros (visited on 05/17/2021).

[29] *Overview of model structure about YOLOv5 · Issue #280 · ultralytics/yolov5*. en. URL: https://github.com/ultralytics/yolov5/issues/280 (visited on 05/18/2021).

[30] Chien-Yao Wang et al. “CSPNet: A New Backbone that can Enhance Learning Capability of CNN”. en. In: *arXiv:1911.11929 [cs]* (Nov. 2019). URL: http://arxiv.org/abs/1911.11929 (visited on 05/18/2021).

[31] Shu Liu et al. “Path Aggregation Network for Instance Segmentation”. en. In: *arXiv:1803.01534 [cs]* (Sept. 2018). URL: http://arxiv.org/abs/1803.01534 (visited on 05/18/2021).

[32] *IntelRealSense/librealsense*. en. URL: https://github.com/IntelRealSense/librealsense (visited on 05/20/2021).

[33] *What Is Camera Calibration? - MATLAB & Simulink - MathWorks Nordic*. URL: https://se.mathworks.com/help/vision/ug/camera-calibration.html (visited on 05/19/2021).

[34] *Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.7 documentation*. URL: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html (visited on 05/19/2021).

[35] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020). Publisher: Springer Science and Business Media LLC, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[36] Tully Foote. “tf: The transform library”. en. In: *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*. Woburn, MA, USA: IEEE, Apr. 2013, pp. 1–6. ISBN: 978-1-4673-6225-2 978-1-4673-6223-8 978-1-4673-6224-5. DOI: 10.1109/TePRA.2013.6556373. URL: http://ieeexplore.ieee.org/document/6556373/ (visited on 05/15/2021).

[37] *navigation/Tutorials/RobotSetup/TF - ROS Wiki*. URL: http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF (visited on 05/15/2021).

[38] *Simulating Jackal — Jackal Tutorials 0.5.4 documentation*. URL: https://www.clearpathrobotics.com/assets/guides/kinetic/jackal/simulation.html (visited on 05/20/2021).

[39] *REP 105 – Coordinate Frames for Mobile Platforms (ROS.org)*. URL: https://www.ros.org/reps/rep-0105.html (visited on 05/20/2021).

[40] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. “Tracking Objects as Points”. en. In: *arXiv:2004.01177 [cs]* (Aug. 2020). URL: http://arxiv.org/abs/2004.01177 (visited on 05/20/2021).

[41] Levon Khachatryan. *lev1khachatryan/Centroid-Based_Object_Tracking*. Apr. 2021. URL: https://github.com/lev1khachatryan/Centroid-Based_Object_Tracking (visited on 05/19/2021).

[42] Rainer E. Burkard and Eranda Çela. “Linear Assignment Problems and Extensions”. en. In: *Handbook of Combinatorial Optimization*. Ed. by Ding-Zhu Du and Panos M. Pardalos. Boston, MA: Springer US, 1999, pp. 75–149. ISBN: 978-1-4419-4813-7 978-1-4757-3023-4. DOI: 10.1007/978-1-4757-3023-4_2. URL: http://link.springer.com/10.1007/978-1-4757-3023-4_2 (visited on 05/20/2021).

[43] Richard W. Bohannan. “Comfortable and maximum walking speed of adults aged 20—79 years: reference values and determinants”. In: *Age and Ageing* 26.1 (1997), pp. 15–19. ISSN: 0002-0729. DOI: 10.1093/ageing/26.1.15. URL: https://doi.org/10.1093/ageing/26.1.15.

[44] Abduallah Mohamed et al. “Social-STGCNN: A Social Spatio-Temporal Graph Convolutional Neural Network for Human Trajectory Prediction”. en. In: *arXiv:2002.11927 [cs]* (Mar. 2020). URL: http://arxiv.org/abs/2002.11927 (visited on 05/20/2021).

[45] Bharath Gopalakrishnan et al. “Prvo: Probabilistic reciprocal velocity obstacle for multi robot navigation under uncertainty”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1089–1096.

[46] Anders Grunnet-Jepsen, John N Sweetser, and John Woodfill. *Best-Known-Methods for Tuning Intel® RealSense™ D400 Depth Cameras for Best Performance*. en.

[47] *Intel RealSense D400 Series Product Family Datasheet*. en. Feb. 2021. URL: https://dev.intelrealsense.com/docs/intel-realsense-d400-series-product-family-datasheet (visited on 05/16/2021).

[48] *Projectors for D400 Series Depth Cameras*. en. URL: https://dev.intelrealsense.com/docs/projectors (visited on 05/16/2021).

Non-exclusive licence to reproduce thesis and make thesis public

I, Jüri Jõul

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   "3D bounding box detection of moving objects for robot navigation in dynamic environments"

   supervised by Arun Kumar Singh

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Jüri Jõul
20.05.2021