Lukas Reisinger, BSc

Dynamic Obstacle Detection And Dynamic Navigation In Crowded Environments

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Assoc. Prof. Dipl-ing. Dr.techn Gerald Steinbauer

Institute for Software Technology (IST)

Graz, November 2018

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

Abstract

Robotics in general is getting increasingly important and is influencing our everyday live more and more. Autonomous mobile robots are nowadays not only used in separated areas specially dedicated to robots, they are also used in industry and on public streets or areas. Thus, dynamic obstacles like people or other autonomous driving vehicles are moving next to the robot.

In this master thesis, an approach to enable dynamic obstacle avoidance in an already existing navigation stack is presented. Thus, first of all dynamic obstacles have to be detected. This obstacle detection is based on 3D data delivered by a stereo vision system. Later, it is shown that the proposed approach can be also used in combination with data delivered by a 3D range scanner. After all obstacles of the environment are detected and separated from the background, motion information according to the obstacle like position and velocity are estimated with the help of a multi-object tracking system. This tracking system is generally based on a Kalman Filter and a Global Nearest Neighbour data association approach.

The avoidance maneuver itself is generated based on the data delivered by the already existing navigation stack and the new gathered data. The main approach for the dynamic obstacle avoidance used within this thesis is based on the collision cone concept called Velocity Obstacles.

Finally, a detailed evaluation of the navigation system is presented, where the new presented system is compared to the already existing system. Further, a comparison of the navigation systems based on a survey with respect to human factors is also done.

Zusammenfassung

Robotik im Generellen wird heutzutage immer wichtiger und beeinflusst unser tägliches Leben mehr und mehr. Autonome mobile Roboter werden heute nicht nur in speziell abgetrennten Bereichen eingesetzt, sie kommen immer öfters in der Industrie und auch an öffentlichen Bereichen zum Einsatz. Dadurch entstehen neue knifflige Situationen, denn in solchen Bereichen bewegen sich auch andere dynamische Objekte wie zum Beispiel Menschen oder auch andere Roboter.

Im Rahmen dieser Masterarbeit wird ein Konzept präsentiert, das Kollisionen mit dynamischen Objekten verhindert. Das Ziel ist es, diesen neuen Ansatz in ein bereits existierendes Navigationssystem eines Transportroboters zu implementieren.

Damit Kollisionen mit dynamischen Objekten verhindert werden können, müssen diese zuerst erkannt werden. Diese Hinderniserkennung basiert auf 3D-Daten, die von einem Stereo-Vision-System geliefert werden. Später wird gezeigt, dass das präsentierte Konzept auch in Kombination mit Daten, welche von einem 3D Laserscanner geliefert werden, verwendet werden kann. Nachdem alle Hindernisse der Umgebung erkannt und vom Hintergrund getrennt wurden, werden Bewegungsinformationen in Bezug auf Position und Geschwindigkeit mit Hilfe eines Multi-Objekt Tracking-Systems ermittelt. Dieses Tracking-System basiert im Allgemeinen auf dem oft genutzten und sehr bekannten Kalman-Filter.

Das Ausweichmanöver selbst wird durch die Kombination der Daten des bereits existierenden Navigationssystems und den neu ermittelten Daten berechnet. Diese Berechnungen basieren hauptsächlich auf dem Konzept von sogenannten Collision-Cones oder besser bekannt als Velocity Obstacles.

Abschließend wird eine detaillierte Auswertung des Navigationssystems präsentiert, wobei das neu vorgestellte System mit dem bereits bestehenden System verglichen wird. Des Weiteren wird auch ein Vergleich der Navigationssysteme basierend auf einer Umfrage in Bezug auf menschliche Faktoren durchgeführt.

Contents

Chapter 1

Introduction

This master thesis focuses on autonomous navigation of a mobile robot in crowded dynamic environments. Nowadays, this topic is of great interest and a lot of effort is spent on its research. Autonomous navigation in crowded environments is mainly based on the three pillars obstacle detection, dynamic obstacle tracking, and autonomous navigation considering these dynamic obstacles. All these topics has their own challenges. There are existing several different approaches, methods and sensor settings to tackle each of them.

To limit the scope of this work and to set some boundaries, there is the constraint to fulfil the task with a stereo vision system mounted on a mobile wheel-based platform, where the localization stack is already provided by this platform.

In the further course of this chapter, the outline and a more detailed problem description can be found.

1.1 Outline

In the next section of this chapter a detailed description of the initial situation will be given followed by the presentation of the aim of this work.

The Prerequisites chapter captures all the theoretical background which is needed to follow and understand the methods described and used later on within this thesis.

The next chapter after Prerequisites is called Related Research and is therefore focusing on previous published work which is addressing either exactly the same task, kind of related problems or only sub-problems of this thesis.

Afterwards there will be a detailed description of the overall system and its implementation, followed by the results of the empirical evaluation and the final conclusion and possible improvements for the future.

1.2 Initial situation

Robotics in general is getting increasingly important and is more and more influencing our everyday live. Autonomous mobile robots are nowadays not only used in separated areas specially dedicated to robots, they are also used in industry and on public streets or areas. In the industry autonomous mobile robots are mostly used as a mean of transport, to carry goods, products or other objects within warehouses or manufacturing facilities.

As a result, a crucial ability of today's mobile robots is to operate autonomously in complex environments [59]. In such environments persons and other kind of dynamic obstacles, such as forklifts are moving around in close proximity to the robot. In warehouses or production sites, like in most public places, this scenario is often worsened by narrow corridors or areas crowded with randomly moving people.

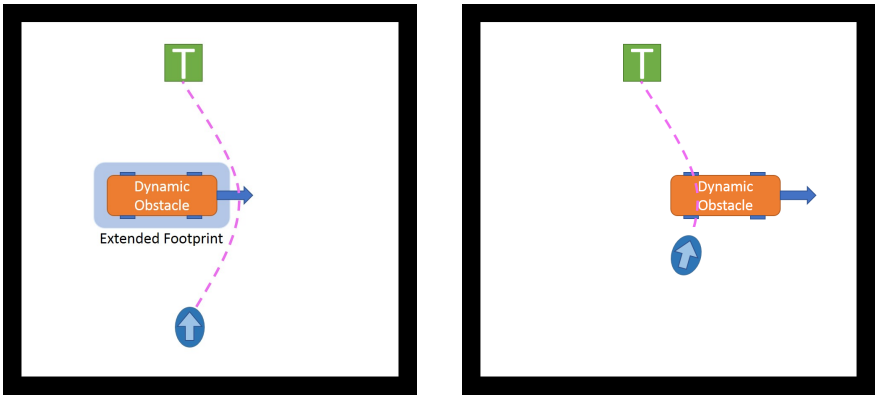
The primary goal of a mobile robot is to autonomously find and execute a path to a given target without colliding with any kind of obstacles. Moreover, the calculated path should be the shortest path possible.

The goal of this work presented in this thesis is to improve an already existing navigation system, by using information derived from dynamic obstacles. In the current situation all dynamic obstacles are almost handled the same way as static obstacles. A simple approach to overcome collisions with dynamic obstacles is to tread all dynamic obstacles as static once and to expend the footprint of dynamic obstacles by a certain safety distance. This is exactly the way, how the already existing system deals with dynamic obstacles.

The above described approach enables obstacle avoidance but brings up new problems. In highly crowded environments with moving obstacles, as for example, there can be the situation that no path to a certain target can be found, because of the massive footprint expansion. There might occur also the situation, that a calculated path is not going to be the real shortest path, since the expansion is always a crude assumption.

Summarized this means, dynamic obstacles are considered statically in path planning, but velocity and possible future positions of dynamic obstacles are not considered.

A further possible scenario is shown in Figure 1.1a and 1.1b.



(a) The robot is planning to pass the dynamic obstacle on the right.

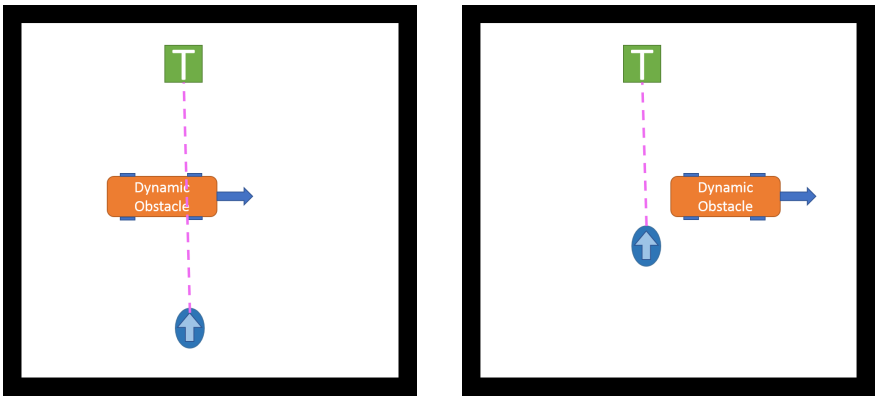
(b) The dynamic obstacle is blocking the path of the robot.

Figure 1.1: Dynamic obstacle is considered as static during the planing phase.

In this situation, the dynamic obstacle is considered as a static once. The robot wants to calculate and execute a shortest path from its current position to the target, illustrated as green box and labelled with T, while a dynamic obstacle is moving from the left side of the image, to the right side.

At time frame t_1 (Figure 1.1a), the robot is planning to pass the obstacle on the right. At time frame t_2 (Figure 1.1b), the robot is going to execute the previously calculated path, but at this point in time the dynamic obstacle was already moving further. Now the dynamic obstacle is crossing the path of the robot and the robot has to manage this situation.

This scenario can be easily avoided by taking into account the information of velocities and future positions of dynamic obstacles, as shown in Figure 1.2a and 1.2b.



(a) Dynamic path planning.

(b) Robot is passing the obstacle ideally.

Figure 1.2: Future position of the dynamic obstacle is considered in the path planing phase.

From the example above we can conclude, that current and future movements of dynamic obstacles have to be estimated and considered in the path planning and execution to ensure efficient and safe path planning and execution.

1.3 Problem description

The movement prediction of dynamic obstacles and the subsequent planning of the path in narrow corridors or crowded areas is a non-trivial task for mobile robots. Thus, this particular problem should be tackled within this thesis, to allow safe autonomous navigation in dynamic environments.

To enable autonomous navigation of mobile robots on environments as described above reliable data from sensors is needed. Currently, very often 2D data from laser range scanners is used, since the delivered data usually has a very high accuracy. However, nowadays 3D data, delivered by a stereo vision system for example, is being used more often. The advantage of such stereo cameras lies in the fact that in addition to the accurate depth information, these sensors also provide color information that can be used as well.

Both the depth information as well as the color information, provided by the stereo camera, can be used to separate essential obstacles, as for example people or other moving obstacles, from non-essential obstacles, such as the ground, walls, shelfs or other static obstacles belonging to the environment. This is important, because we are especially interested in obstacles in the foreground and dynamic obstacles. With the help of the provided 3D information, obstacles can be identified and new information as for example moving direction or velocity of the obstacle can be gathered.

For a stable navigation in narrow and crowded areas, the current motion and the estimation of the future motion of dynamic obstacles is a key factor. Especially the estimated future velocity and direction of dynamic obstacles needs to be considered in the navigation.

Dynamic obstacles and all the information according to these obstacles can be estimated, by evaluating continuous data given by the stereo vision system over a longer period of time. The difficulty here is to filter the data properly to reduce noise, get the right data associations, and to find a suitable motion model to accurately predict future motion sequences. People for example can suddenly change their direction of movement by 180 degrees, while forklifts must drive a curve with a certain radius or initiate a breaking process before they can change direction by 180 degrees. Thus, the motion models of people and forklifts are different. Since the goal of this thesis is to use the information of dynamic obstacles for the navigation in general, no object classification is applied on the detected obstacles. Thus, a reliable generalized motion model, which is assigning best and able to describe most of the possible motions of various dynamic obstacles has to be chosen as foundation for the further tracking process.

A correct data association, as mentioned above, is also a major challenge tackled in this thesis. Data association is about all the current detections to the previously estimated predictions to increase the accuracy of further estimations. As a consequence of this, a wrong association of detections and estimations will

lead to inaccurate or completely wrong motion estimation and thus also to an unreliable navigation.

1.4 Contribution of the thesis

The overall goal of this thesis is to allow safe autonomous navigation of a mobile robot in dynamic environments. Therefore, static as well as dynamic obstacles in the environment have to be detected correctly. The position of the detected obstacles is later passed to an obstacle tracking algorithm to obtain motion information. Due to the motion information, possible future positions of dynamic obstacles can be estimated. These estimations are finally passed to the navigation unit and used to find and execute a path to a specific target which is short and do not cause any collisions.

Beside the technical aspects there are also a number of social aspects related to autonomous mobile robots, which are interacting with people in a close way. Persons which are directly next to the mobile robot should not feel scared and disturbed by the navigation behavior of the robot.

However, the feeling of individual persons cannot be determined quantitatively in the first place. Nevertheless, there is the possibility to interview people in a test scenario and perform statistic evaluations on the gathered data. Furthermore, through these evaluations different navigation approaches can be compared, in respect to the acceptance by people.

Additionally, the overall concept can be quantitatively evaluated in relation to parameters such as throughput over a certain period of time or the covered distance to reach a specific target.

Chapter 2

Prerequisites

The following sections of this chapter provide all the knowledge that is needed to understand this thesis. Thus, this chapter includes all the technical as well as the theoretical background to comprehend the methods which are discussed or used later on within this work.

First of all, we will give a short description of the basic functionality of the commonly used Robot Operating System or short ROS [58]. ROS is providing the fundamental infrastructure for the practical realization described in this thesis.

This description is followed by an explanation about stereo vision in general and further vision and stereo vision related topics as for example disparity maps.

This chapter does also contain a summary and explanations of specific methods of the well known Point Cloud Library (PCL) [62]. Several methods provided by this library are later used to perform certain preprocessing tasks.

At the end of this chapter a detailed description of the Kalman Filter, followed by an explanation of different data association approaches can be found.

2.1 Robot Operating System

The Robot Operating System was originally introduced in 2007 by the Stanford Artificial Intelligence Laboratory [58]. In this section a general overview of the main functionality of ROS is provided.

As mentioned in [58], the Robot Operating System is not an operating system in the traditional sense providing for instance process management and scheduling. It rather provides a structured communication layer above the host operating system. Furthermore this system makes it possible to distribute the computations of different units in a very easy way to several machines.

Nowadays, ROS is not only a communication framework anymore. It provides a lot of drivers for different sensors, various tools and many libraries. A very big benefit of the Robot Operating System lies within the fact, that it is multi lingual. This simple means that C++ as well as Java or Python can be used to program robots. Furthermore, there is a big community which is using

and creating open source software based on ROS which can be easily reused by all the ROS users all over the world [48].

In the next section the general structure of the Robot Operating System will be presented, followed by an explanation of some basic ROS tools.

2.1.1 Packages

ROS is following a really strict structure to make software easy exchangeable, reusable, and expandable [58].

The Robot Operating system is structured in so called packages. Packages can be added, removed, or changed independently. Such packages mostly cover a broader range of functionalities as for example everything that is needed to run a simple web_cam-driver on the robot. It is recommended to follow a specific package structure which is schematically shown in Figure 2.1

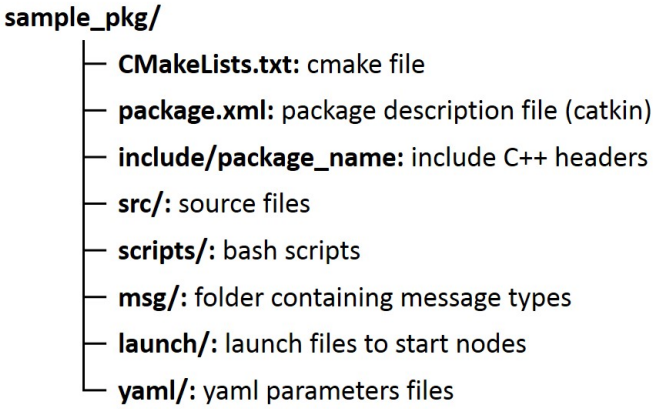


Figure 2.1: Standard ROS package structure.

In general, a package can contain one or more executables and libraries.

2.1.2 Nodes, Topics and Messages

An executable itself is called node, where a node is a process that performs certain computations. A full robot control system comprises many different nodes. For example, one node is used as a driver for a laser range finder and another one is used to control the robot wheels. The computation unit is called node, since the system structure and the communication between several units can be represented as a graph.

The individual nodes can communicate with each other via so called topics. In the perspective of a graph, such topics can be represented as edges connecting different nodes. At this point there has to be mentioned, that there can be more than one edge between two nodes. Multiple connections are unavoidable since a single topic can only transport a specific message type. An illustration of a small sample robot system is shown in the Figure 2.2 below.

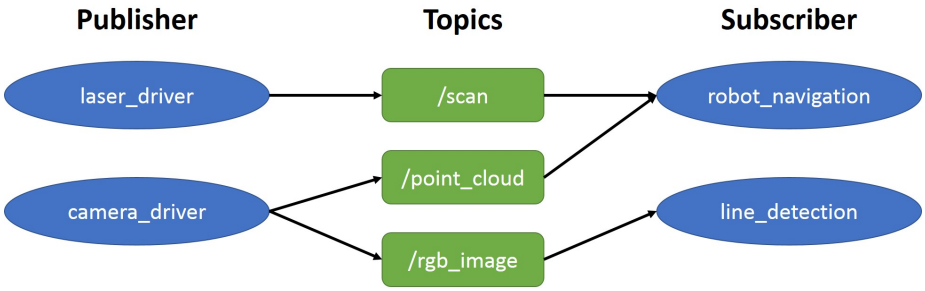


Figure 2.2: Simple ROS robot system.

As mentioned above, nodes can communicate with each other, by sending messages of a certain type to a topic. This communication is asynchronous. If a node A wants to receive a message from another node B, node A has to subscribe to the specific topic. Therefore node A is also called subscriber.

Node B on the other hand has to publish messages to the same topic and is therefore called publisher. The message type itself has to be defined at compile time, where the standard message of a laser range finder provided by basic ROS libraries is shown in Listing 2.1.

```
// sensor_msgs/LaserScan.msg

Header header
float32 angle_min        // start angle of the scan [rad]
float32 angle_max        // end angle of the scan [rad]
float32 angle_increment  // angular increment [rad]
float32 time_increment   // time between measurements [s]
float32 scan_time        // time between scans [s]
float32 range_min        // min scan range [m]
float32 range_max        // max scan range [m]
float32[] ranges         // range data [m]
float32[] intensities    // intensity data []
```

Lst. 2.1: Laser scan message definition.

In respect to the message defined above, the header field itself is also a message. The definition of the header message is shown in Listing 2.2.

```
// std_msgs/Header.msg

uint32 seq               // sequence number
time stamp               // time stamp
string frame_id          // coordinate frame
```

Lst. 2.2: ROS header message definition.

Thus, messages can be capsulated, which is absolutely fitting the purpose of software reuse.

2.1.3 Parameters

In the field of robotics, algorithms which can be individually adapted and fine-tuned to certain environments are frequently used to increase the overall performance of the robot for certain scenarios. A simple reference algorithm is for example a range filter for the data delivered by a laser range finder. At environment A the maximum range of the laser scanner should be restricted to 30 meters, while at environment B or for testing reasons the maximum range should be only 15 meters. The fundamentals of the range filter are staying the same, but the threshold for the range is changing from 30 to 15 meters. With the infrastructure provided by ROS, this adaption can be easily done without recompiling the whole code. ROS is providing a parameter server where such thresholds and many other parameters can be managed and also changed during runtime.

The current parameter set can be retrieved with the terminal command:

```
rosparam list
```

, where a specific parameter can be set or retrieved with the commands:

```
rosparam get <parameter name>
rosparam set <parameter name> <value>
```

2.1.4 ROS Master

The most important component of the whole Robot Operating System is the so called ROS Master. The ROS Master is the heart of the Robot Operating System and is providing naming and registration services for all the nodes in the system and enables the communication between the nodes. Without the ROS Master the nodes cannot be started. Thus, the first thing which has to be done to start a robot is to initialize a ROS Master. This can be simple done with the terminal command:

```
roscore
```

2.1.5 Transformations

In this section of the chapter, the ROS coordinate system structure and transformations of positions between the individual coordinate systems are described.

A robot system with all its sensors and actors can be represented by several coordinate systems. Usually each sensor and each actor has its own coordinate system. Furthermore, coordinate systems can be introduced to ease calculations or to increase understandings. Note that in ROS usually the coordinate systems are handled as right hand coordinate systems as illustrated below in Figure 2.3. Thus, the x-axis is pointing forward, the y-axis is pointing to the left and the z-axis is pointing upwards.

Coordinate systems in ROS are called frames and all the frames are organized in a tree. A sample transformation tree of a simple wheel based robot using a laser range scanner and a camera is presented in Figure 2.4 and 2.5.

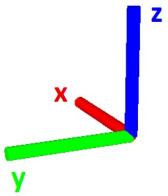


Figure 2.3: Right handed coordinate system.

In the Figure 2.4 the physical placements of the different robot frames are illustrated. The **world** frame is representing a fixed reference frame, while the **base_link** is representing the center of the robot. **laser_link** and **camera_link** are representing the pose of the sensor centers with respect to the **world** frame. In this example it is easy to see, that the camera is mounted exactly above the laser scanner. The **odom** frame is representing the odometry and shows the initial position of the robot.

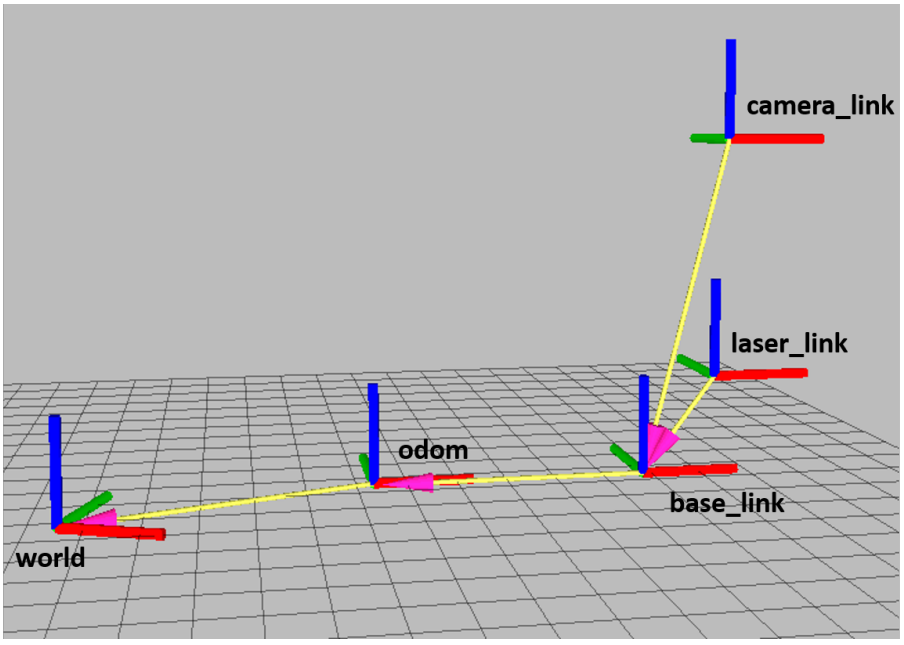


Figure 2.4: tf-tree representing the positions and orientations of all frames of a sample robot system.

Data delivered by any kind of sensor is mostly related to a specific sensor coordinate system. In the example above, this would be the **laser_link** frame or the **camera_link** frame. To use these points within other frames, as for example within the **world** frame, all the data points has to be transformed to the desired frame. Therefore, all transformations shown in the tf tree of Figure 2.5 between **laser_link** and **world** has to applied to the data points.

Transforming points between coordinate systems is not a thing which is

happening once in a while. Such transformation calculations are usually done several times a second. Since it is pretty hard to deal with the transformations on your own, ROS is providing a package called **tf** which provides coordinate frame transformations. The **tf** package is simple to use and makes working with transformations easier, understandable and safer.

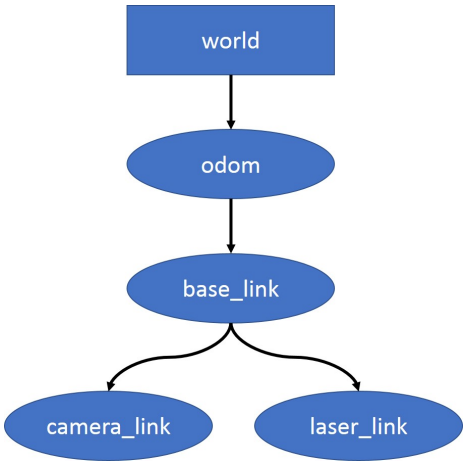


Figure 2.5: tf-tree

2.2 Vision and Stereo Vision

In this section and the following subsections basic knowledge about stereo vision and computer vision methods in general is provided. Since the topics computer vision and stereo vision are really broad topics, the focus lies strict on providing background information for methods directly used or discussed within this thesis.

Therefore depth reconstruction based on triangulation, disparity maps, and Hough lines are explained in the following paragraphs.

2.2.1 Triangulation

Stereo systems and stereo vision are generally used within the field of robotics to get 3D information of a certain scene. When using a single pinhole camera as a sensor, useful 3D information of the scenery is getting lost, since 3D world points are projected onto an 2D image plane. This problem can be overcome by using multiple cameras or images and combining the information of both images with the help of epipolar geometry and triangulation [72].

For further descriptions we set up the constraint, that the two images are given by two pinhole cameras which are located next to each other. Thus the data is gathered from a simple stereo vision system. The extrinsic as well as intrinsic parameters of the camera are well known. The pinhole camera model as well as the stereo vision system are illustrated at the Figures below.

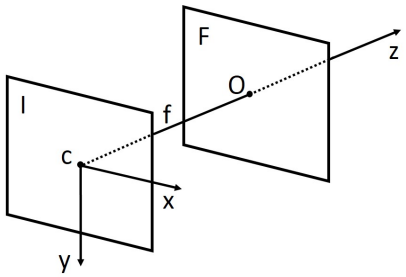


Figure 2.6: Simple pinhole camera model.

In Figure 2.6, a pinhole camera model is presented, where I is the image plane and F is the focal plane. C is the center of the image plane and also called principal point, O is the optical center and f denotes the focal length which is exactly the distance measured between the image plane I and the focal plane F. The focal length f is either given in millimeters or in pixels.

The main goal of the above mentioned epipolar geometry is to find corresponding image points within the left and the right image plane, which are representing the projection of a 3D point in the scenery. Some algorithms to detect these correspondences are for example described in [74], but will be not captured here in detail. For the triangulation and the disparity and depth calculation it is assumed that these correspondences are already known.

The goal of the triangulation process is to reconstruct the 3D point P of the scenery, given two corresponding points x_L and x_R on the image planes I_L and I_R. For the following calculation it is assumed that the cameras are level and the image planes are flat and co-planar as illustrated at Figure 2.7. With respect to Figure 2.7 O_L and O_R are representing the optical centers, C_L and C_R are the center points of the image planes, f is the focal length and Z is the distance (depth) between the cameras and the point$P$.

Following the simple mathematical rules of triangulation, the disparity d, which is defined as the difference in image location of the same 3D point, can be retrieved by:

$$d = \overline{X_L C_L} + \overline{C_R X_R} \tag{2.1}$$

Rewritten this can be shown as:

$$d = \left(\frac{\overline{X_L C_L}}{\overline{O_L C_L}} + \frac{\overline{C_R X_R}}{\overline{O_L C_L}} \right) \cdot \overline{O_L C_L} \tag{2.2}$$

$$d = \left(\frac{\overline{X_L C_L}}{\overline{O_L C_L}} + \frac{\overline{C_R X_R}}{\overline{O_R C_R}} \right) \cdot \overline{O_L C_L} \tag{2.3}$$

Applying the rules of triangulation, the equation above can be changed to:

$$d = \left(\frac{\overline{O_L C}}{\overline{PC}} + \frac{\overline{C O_R}}{\overline{PC}} \right) \cdot \overline{O_L C_L} \tag{2.4}$$

$$d = \frac{\overline{O_L O_R}}{\overline{PC}} \cdot \overline{O_L C_L} \tag{2.5}$$

Considering the fact that $\overline{PC}$ is nothing else than the depth Z, $\overline{O_L O_R}$ is the so called base line b (distance between the two camera centers) and $\overline{O_L C_L}$ is the focal length f, the formula can also be written as

$$d = \frac{b \cdot f}{Z} \tag{2.6}$$

and the depth Z can be retrieved by:

$$Z = \frac{b \cdot f}{d} \tag{2.7}$$

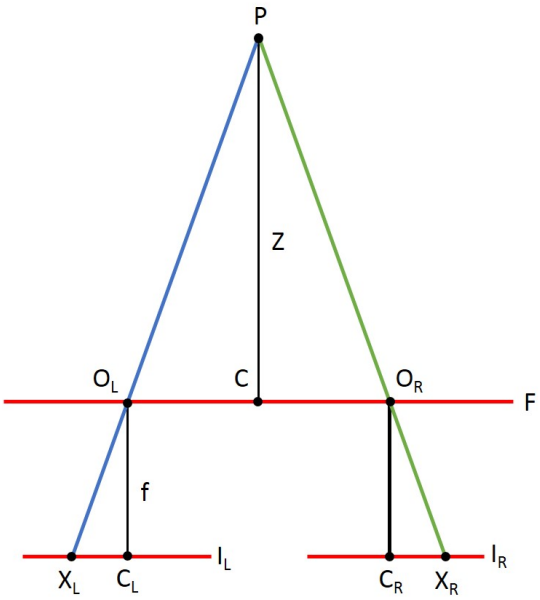


Figure 2.7: Stereo camera triangulation.

2.2.2 Disparity Maps

In the section above the reconstruction of a single point is explained. In reality, of course the disparity for every pixel pair is calculated to reconstruct the depth. These results are later on stored in so called disparity maps. A disparity map can be represented, as well as the image plane, as a matrix. If we assume that the size of the left image plane I_L and the size of the right image plane I_R are equal, also the disparity map is of the same size. Thus if I_L is of size w x h, where w is the image width and h is the image height, also the disparity map will have the same size w x h. The value of a single cell of the disparity map is representing the disparity calculated with the help of Equation 2.6 presented in section 2.2.1.

In computer vision such a disparity map is often represented as a gray scaled image, where the disparity values are normalized to the range of 0 to 255 or 0 to 1. A sample image of such a disparity map resulting from two images given by a stereo vision system is shown in the Figure 2.8 below.

(a) RGB reference image. (b) Gray scaled disparity image.

Figure 2.8: Disparity image of a person in a production hall.

Disparity maps in general are often used as input data for further steps as for example object detection or ground detection based on U-V disparity [49] or simple for depth map generation.

2.2.3 Hough Transformation

In this section, the basics of the Hough transformation, especially in relation to line detection are described.

In the field of computer vision Hough transformations are often used to detect simple shapes, such as straight lines or circles in images as presented in [15]. The input image to the transformation algorithm is in general an image pre-processed by any edge detection algorithm as for example the canny edge detector presented in [12].

The simplest case for which Hough transformations can be used, is to detect straight lines. A straight line in the Cartesian space can be represented by:

$$y = k \cdot x + d \tag{2.8}$$

where k is the slope, d is the y-intercept and x is a variable.

Since a line, which can be also seen as a collection of points, is harder to manage than a single point, the goal of the Hough transform is to transform a line into a point, without loosing any information. This can be fulfilled by mapping the line into another parameter or feature space which is here called k-d space, but often also called Hough space. Thus the line $y = kx + d$ in Cartesian space can be represented as point (k, d) in the k-d parameter space as illustrated at Figure 2.9.

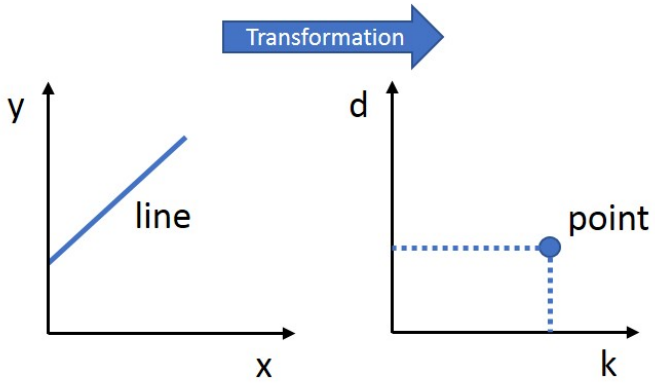


Figure 2.9: xy-kd space transformation.

Note, that based on this notation, there are occurring computational problems when vertical lines are processed. Therefore in [15] it is proposed to use the Hesse normal form:

$$r = x \cdot cos\theta + y \cdot sin\theta \tag{2.9}$$

, where ρ is the distance from the origin to the closest point of the line and θ is the angle between the x-axis and ρ as shown in Figure 2.10.

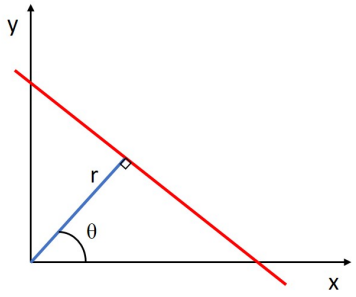


Figure 2.10: Hesse normal form representation.

The line equation shown in Equation 2.8 can be also written as:

$$y = \left(-\frac{cos\theta}{sin\theta}\right) \cdot x + \left(\frac{r}{sin\theta}\right) \tag{2.10}$$

,where θ has to be defined to be in the range of $[0, 2\pi[$ and $r > 0$.

An algorithm for Hough line detection is presented below based on the descriptions in [15]:

```
for all pixels p in the image:
if p at (x,y) is an edge pixel:
for all theta in theta[]:
rho = calcRho(x,v,theta)
accumulator.at(rho, theta).increment()
```

Lst. 2.3: Hough line detection.

Having a pixel p in an image with the position (x, y), infinity many lines can go through the pixel p. All this lines can be transformed to the Hough space, which will finally lead to a sinusoidal curve that is unique for pixel p. The same procedure has to be done for all the other edge pixels of the image. The point of intersection between two or more resulting sinusoidal curves in the Hough space is representing a line in the image space.

The result of all this transformations is stored in a so called accumulator, which can be simple represented as a matrix, where cells with sinusoidal curve intersections have a higher value than others. After all points of interest are transformed from the Cartesian space to the Hough space, local maxima in the accumulator are searched. This is usually done with the help of simple thresholding. Based on Equation 2.10, all found local maxima are mapped from the Hough space to the Cartesian space as illustrated in Figure 2.11.

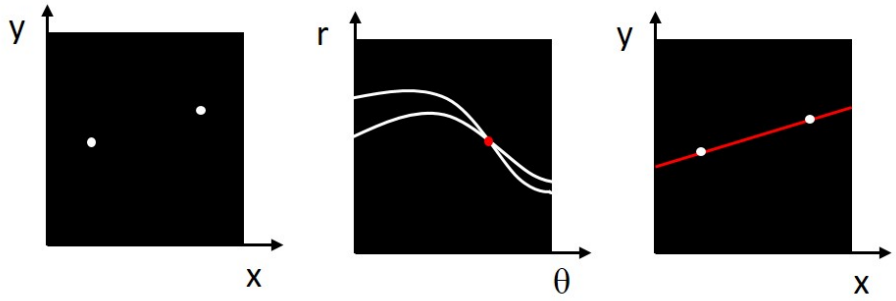


Figure 2.11: Hough line detection.

2.3 Point Cloud Library

In this section of the thesis there can be found a general overview of the well known Point Cloud Library (PCL). This overview is followed by some basic description of algorithms, which are later used for the practical realization.

As described in [62] and at PCL [1], PCL became a first-class citizen project in March 2011 and contains all the state-of-the-art algorithms related to 3D perception. The main thematic areas are:

- filtering
- feature estimation
- surface reconstruction
- registration
- model fitting
- segmentation

Thus, the application area of this library is pretty large, but the following descriptions are just touching the area of filtering, since for this thesis PCL is only used for data preprocessing. Exactly, the basic point cloud container, voxel grids as well as the statistical outlier removal algorithm are described below.

2.3.1 Point Cloud

A point cloud is the fundamental data container of the PCL library. Such a point cloud consists either of 2D points or 3D points. This special data structure makes it possible to easily access single points or even single point coordinates. Every method and algorithm which is implemented in PCL is based on this structure.

A short example how elements and single coordinates of a point cloud can be accessed is shown below:

```
// create a 3d point cloud
pcl::PointCloud<pcl::PointXYZ> cloud;

// create a new 3d point
pcl::PointXYZ point;

// assign values to x, y and z coordinates of the point
point.x = 45.12
point.y = 2.123
point.z = 321.08

// add the point to the point cloud
cloud.points.push_back(point);
```

Lst. 2.4: Point cloud container.

[1] http://pointclouds.org, 02.10.2018

2.3.2 Voxel Grid

In this section, the functionality of voxel grids is described in a few sentences.

Voxel grids are nowadays used within many different applications as for example for 3D map creation (OctoMaps) [26], but also as data filter for following object detection algorithms [64].

The name voxel is a combination of the two words volume (vo) and element (el). A single voxel can be represented by a cuboid, where this cuboid is defined by a certain width, height and depth as illustrated below.

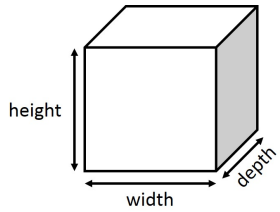


Figure 2.12: Single voxel.

A voxel grid is simple a inter linkage of many voxels as shown in Figure 2.13. Note, that empty voxels of the grid are colored white and are not visible in the image below. In Figure 2.13 on the left side there can be seen a voxel representation of a tree and on the right side there is shown a simple cube.

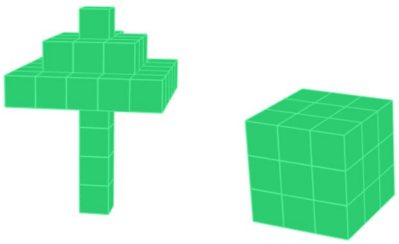


Figure 2.13: Objects represented as voxels.

Applying a voxel grid filter to a point cloud in the sense of PCL means, that all points of the point cloud within a single voxel are combined. Combined in this case means, that the center of gravity, often also called centroid, of all the points in the voxel is calculated and added to a new cloud. Thus the number of points of the point cloud is reduced.

Note, that the centroid c can be calculated with the following formula:

$$c = \frac{\sum_{p \in P} p}{N} \tag{2.11}$$

, where c and p are points in vector form $(x, y, z)^T$, P is the family of all points of a point cloud and N is the number of points within P.

Bellow there can be found a short example, which illustrates the use of a voxel grid filter:

```
// initialize input cloud
pcl::PointCloud<pcl::PointXYZ> input_cloud;
// ... ... load some data

// create container for output cloud
pcl::PointCloud<pcl::PointXYZ> filtered_cloud;

// create voxel grid object
pcl::VoxelGrid<pcl::PointXYZ> voxel_grid_filter;

// set the size of a single voxel
double width = 0.1;
double height = 0.1;
double depth = 0.1;
voxel_grid_filter.setLeafSize (width, height, depth);

// additionally set the minimum number of points
// which have to be in a voxel
int min_points = 3;
voxel_grid_filter.setMinimumPointsNumberPerVoxel(min_points);

// apply the filter to the input cloud
voxel_grid_filter.setInputCloud(input_cloud);
voxel_grid_filter.filter(filtered_cloud);
```

Lst. 2.5: Voxel grid filter in PCL.

Further, in the PCL implementation of the voxel grid filter there is the possibility to set the minimum number of points which has to be within a voxel to consider the voxel as valid. This functionality can be used to reduce the noise introduced by the sensor.

2.3.3 Statistical Outlier Removal

In this section a brief description of the statistical outlier removal algorithm presented at the PCL tutorial [2] is given.

In general, measurement errors of sensors as for example a stereo camera or a 3D range finder, can lead to sparse outliers. These outliers can corrupt the later results of other algorithms applied to the data, since the local point cloud characteristics such as surface normals or curvature changes consists of erroneous values. The appearance of outliers can be partly reduced by performing a statistical analysis and remove points which do not meet a certain criterion.

The statistical outlier removal method presented at the PCL tutorial, is based on the distribution of point to neighbors distances, where the Euclidean distance d between two 3D points $p_1(x, y, z)$ and $p_2(x, y, z)$ can be calculated by:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2.12}$$

For each point in the point cloud, the mean distance between the point and its neighbors is calculated, where the number of neighbors considered for this

[2] http://pointclouds.org/documentation/tutorials/statistical_outlier.php, 02.10.2018

process can be defined beforehand. Finally, it is assumed, that the resulted distribution is forming a Gaussian distribution with a mean and a standard deviation. All points whose mean distances are outside a predefined interval are considered as outliers and removed from the dataset.

A sample code, how the statistical outlier removal method given by PCL can be used is listed below:

```
// initialize input cloud
pcl::PointCloud<pcl::PointXYZ> input_cloud;
// .
// .
// ... load some data to input_cloud

// create container for output cloud
pcl::PointCloud<pcl::PointXYZ> filtered_cloud;

// create statistical outlier removal object
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> stat_outlier_filter;

// set the number of neighbors which should be included in
// the process
int number_of_neighbors = 50;
stat_outlier_filter.setMeanK(number_of_neighbors);


// set a threshold in form of multiplier/weight to
// except / not except points
int std_multiplier = 1.1;
stat_outlier_filter.setStddevMulThresh(std_multiplier);

// apply the filter to the input cloud
stat_outlier_filter.setInputCloud(input_cloud);
stat_outlier_filter.filter(filtered_cloud);
```

Lst. 2.6: Statistical outlier removal in PCL.

The results of the statistical outlier removal method applied on a dataset generated by a stereo vision system are illustrated at Figure 2.14. The first picture in Figure 2.14 shows the initial situation, in the picture noisy section are highlighted. The picture at the bottom is representing the result after applying the PCL statistical outlier removal method.

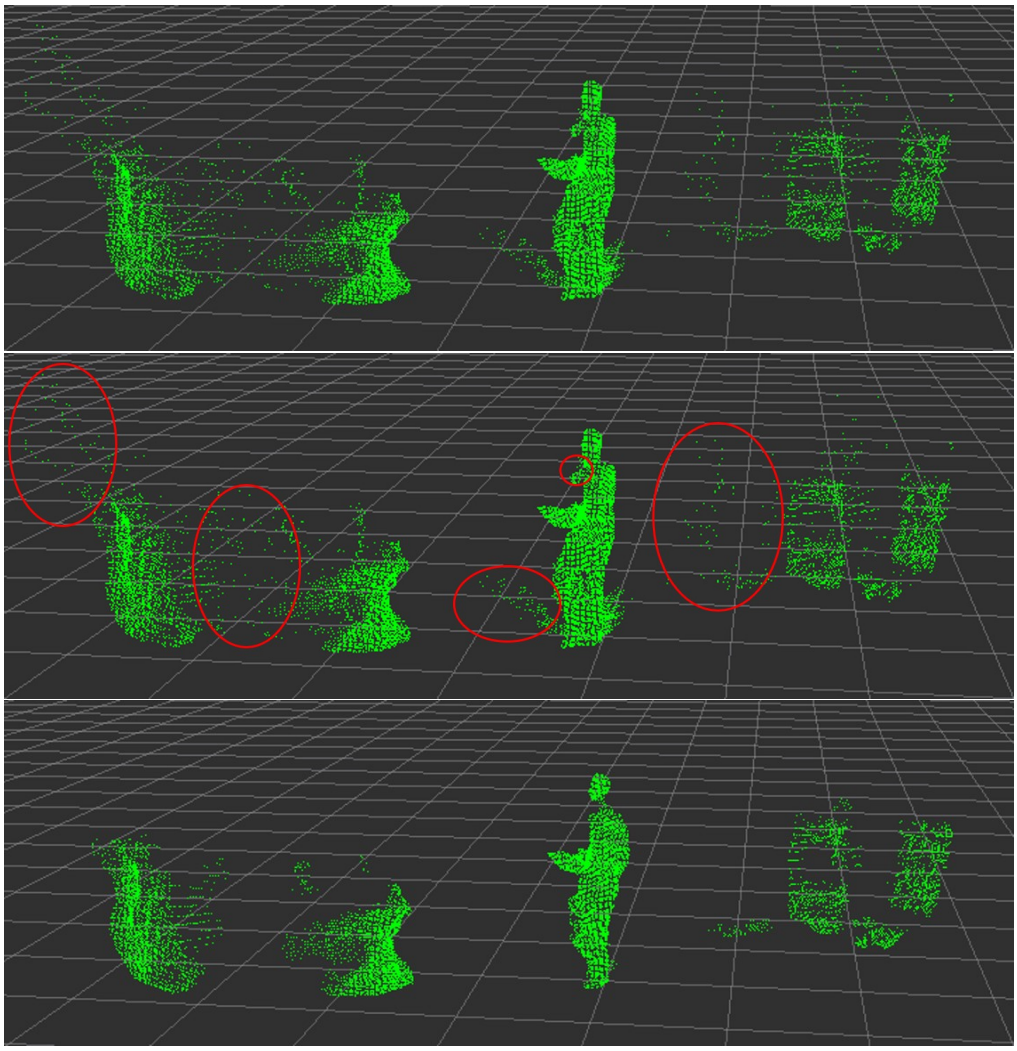


Figure 2.14: PCL statistical outlier removal applied on noisy data.

2.4 Kalman Filter

The Kalman Filter is the classical state estimation method for obstacle tracking and probably one of the most used methods within this context. In the past it was used within all kind of tracking scenarios as for example pedestrian tracking [35], cyclist tracking [37] but also for other obstacles as shown in [8]. The wide adaption is attributable to its simplicity and to the easy implementation.

In the following paragraphs, basic steps and equations for tracking dynamic obstacles with a Kalman Filter are described.

Basically, the Kalman Filter consists of two steps:

- **Prediction step:**

  At this step a prediction of the new state of an obstacle, based on the output of the previous update step is calculated.

- **Update step:**

  The update step tries to take the predicted state and correct it with an observation, to improve further predictions.

As shown in [43] a state and an observation can be noted as x and Z and the two general equations for the state prediction and the current observation transformation can be described as:

$$\hat{x}_k = A \cdot \hat{x}_{k-1} \cdot Bu_{k-1} + w_k \tag{2.13}$$

$$Z_k = H \cdot \hat{x}_k + v_k \tag{2.14}$$

, where the matrix B is only needed to transform data given by the motion control u to the state x and H is only needed to transform a state x to an observation Z. For the purpose of dynamic obstacle tracking, the matrix B and the motion control u are not needed.

w_k and v_k are Gaussian white noises for the process itself and for the observation and will be considered in the noise covariance matrices R and Q. This two matrices can be calculated, as described in [8] by using the variance σ^2 multiplied by the identity matrix I:

$$R = I \cdot \sigma_v^2 \tag{2.15}$$

$$Q = I \cdot \sigma_w^2 \tag{2.16}$$

The matrix A is called transition matrix and is used to relate the state change to a certain motion model. To keep it simple, the equations described in the following are related to a 2D position with Cartesian coordinates (x, y) of an obstacle.

Probably the most used motion model for arbitrary obstacle tracking is the so called constant velocity motion model described in [36] Adapted to a 2D position, the motion model looks like Equation 2.17, where Δt is representing the time difference between two consecutive states.

$$A = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.17}$$

The state $\hat{x}_k$ belonging to the transition matrix A is defined in Equation 2.18.

$$\hat{x}_k = \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} \tag{2.18}$$

where x and y are representing the position of an obstacle and v_x respectively v_y are representing velocities in x and y axis.

In the following explanation the state for the prediction will be written as $\hat{x}$ and the observation will be represented as Z. All the matrices and equations shown below are based on the description of [43].

2.4.1 Prediction

At the prediction step there have to be predicted:

- the **future state** $\hat{x}^-$ by using the previous state $\hat{x}_{k-1}$ and the transition matrix A

$$\hat{x}_k^- = A \cdot \hat{x}_{k-1} \tag{2.19}$$

- the **future state covariance matrix** P_k^- with the help of the transition matrix A, the previous state covariance matrix P_{k-1} and the observation noise covariance matrix Q

$$P_k^- = A \cdot P_{k-1} \cdot A^T + Q \tag{2.20}$$

2.4.2 Update

After the prediction step has been done, there will be the update step, which is in the literature often also called correction step. At this point a measurement is taken to correct or update an associated prediction. At first there has to be calculated the Kalman gain K_k. This matrix is used to weight the innovation and later on it is needed for the correction of the estimated state $\hat{x}$ and the estimated state covariance matrix P_k^-.

$$K_k = P_k^- \cdot H^T (H \cdot P_k^- \cdot H_T + R)^{-1} \tag{2.21}$$

With the help of the Kalman gain from Equation 2.21, the estimation of the predicted state is updated by using the following Equation 2.22

$$\hat{x}_k = \hat{x}_k^- + K_k(Z_k - H \cdot \hat{x}_k^-) \tag{2.22}$$

, where $(Z_k - \hat{x}_k^-)$ or the transformed version $(Z_k - H \cdot \hat{x}_k^-)$ is called measurement innovation and simple describes the difference between the estimation and the observation [43]. The observation vector Z_k is representing the measured position and is defined in Equation 2.23.

$$Z_k = \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.23}$$

and the transformation matrix H is defined in Equation 2.24.

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{2.24}$$

The last step which has to be done is to update the state covariance matrix by using the Kalman gain K_k and the identity matrix I:

$$P_k = (I - K_k \cdot H)P_k^- \tag{2.25}$$

Figure 2.15, the Kalman Filter prediction and update cycle is illustrated.

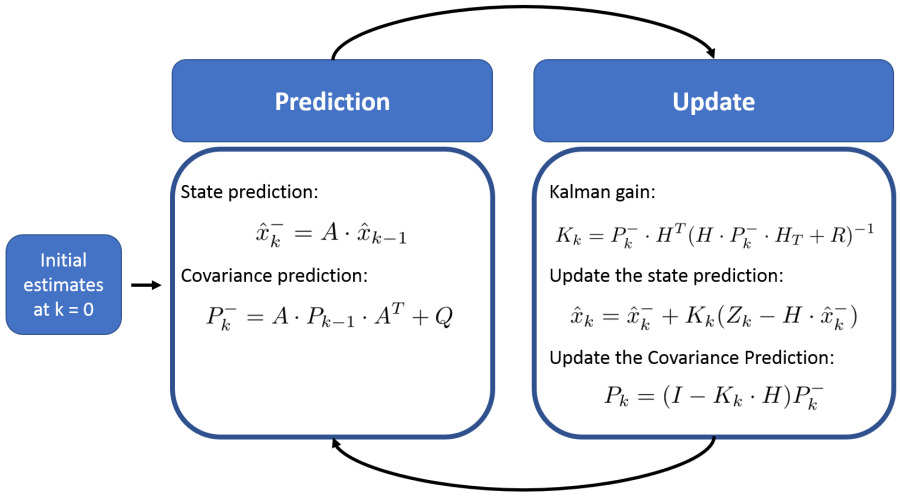


Figure 2.15: Kalman Filter cycle

2.5 Data Association

Data association is a topic which is directly related to tracking. At each update step, a observation has to be associated to a certain prediction. The necessity and difficulty of data association is shown in Figure 2.16.

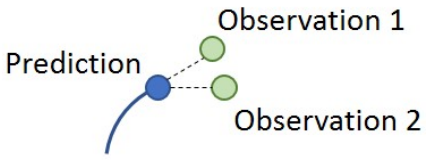


Figure 2.16: Simple data association problem.

In the case presented in Figure 2.16, there are two observations candidates for the prediction update, but there is only one observation really belonging to the prediction.

There are several methods available to solve this assignment problem. Two of them, namely Local Nearest Neighbor Filter and Global Nearest Neighbor Filter, are described below.

2.5.1 Local Nearest Neighbor Filter

The local nearest neighbor filter is one of the easiest approaches which can be used for data association within obstacle tracking.

As described in [8] the nearest neighbor filter simple calculates the distance between all the observations and predictions. At the end of the day, the observation with the minimal distance to the prediction is finally used for the update. Thus, the selected minimum is always a local minimum and this is probably not the best association considering the whole system with multiple observations and predictions.

The distance itself can be represented, as described in [8] by the Mahalanobis distance or also called normalized innovation distance d_k [47]. This distance is calculated by Equation 2.26, using the innovation covariance matrix S_k^{-1} and the innovations v_k given by the Kalman Filter.

$$d_k^2 = v_k^T \cdot S_k^{-1} \cdot v_k \tag{2.26}$$

The innovation v_k can be simple calculated by building the difference between the observation z_k and the prediction state x_k:

$$v_k = z_k - x_k \tag{2.27}$$

With respect to the descriptions in section 2.4 the innovation v_k can be calculated by Equation 2.28.

$$v_k = z_k - H \cdot x_k \tag{2.28}$$

2.5.2 Global Nearest Neighbor Filter

The Global Nearest Neighbor Filter is a kind of improvement of the Local Nearest Neighbor Filter to avoid wrong associations caused by local minima. All the distances between the observations and the predictions are calculated and stored in a so called cost matrix C. This matrix is of size N x M, where N is the number of predictions and M is the number of observations. This cost matrix C can be written, as shown in [30], by:

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1M} \\ c_{21} & c_{12} & \cdots & c_{2M} \\ \vdots & \vdots & \vdots & \vdots \\ c_{N1} & c_{N2} & \cdots & c_{NM} \end{pmatrix} \tag{2.29}$$

, where the cell value corresponds to the Mahalanobis distance based on Equation 2.28. Finally, the optimal association is given by minimizing the total sum overall distances. This can be easily done by using Munkres assignment algorithm presented in [46].

To improve the assignment process a validation gate is introduced, where a maximum distance between an observation and a prediction is defined by the inverse χ^2 cumulative distribution at a significance level α. Thus, if there is only one prediction and one observation and the Mahalanobis distance between them is bigger than the defined threshold, they are not associated.

Chapter 3

Related Research

In this chapter different approaches and methods, which have a strong relation to the topics tackled within this work are described. All the mentioned approaches and methods were already well researched and former published in form of articles, reports or books. The focus in this chapter is on the main idea of the different approaches and the known short comings. Side information and detailed descriptions for sub methods used within the approaches can be found at chapter Prerequisites.

In a very general way the research can be divided into the three main pillars:

- Dynamic obstacle detection
- Dynamic obstacle tracking
- Navigation in crowded dynamic environments

, where each single pillar is more or less a closed topic by itself.

However, dynamic obstacle detection and dynamic obstacle tracking are in the literature often tackled in combination. We will present the different research areas separately.

Since the usage of data delivered by a stereo vision sensor is a constraint for this work, also the research, especially for the dynamic obstacle detection, is limited to approaches and methods based on stereo vision data. Furthermore, there is the assumption, that the disparity map and a xyz-pointcloud are directly provided by the stereo vision system. This further means that stereo matching algorithms will not be discussed within this research.

In the following, each of the above listed pillars is divided into further subareas and different approaches and methods are described.

3.1 Dynamic obstacle detection

In the following sections different approaches and methods for the topic dynamic obstacle detection based on data delivered by a stereo vision system are described.

In the literature dynamic obstacle detection is done by the following approaches:

- extracting obstacles from the environment by using clustering algorithms or obstacle detection algorithms and pass them on to a kind of tracking algorithm
- detecting dynamic pixels directly with the help of optical flow algorithms
- combination of object classification and tracking approaches

All methods have their advantages and disadvantaged; especially with respect to accuracy and computational costs.

The last mentioned approach, tracking by object classification, will not be investigated within this research, since the final results are completely depended on the results delivered by any kind of classification algorithm. Such classification algorithms might be as for example Neural Networks, HAAR classifiers or Histogram oriented gradient classifiers as it is described in [17] and [16]. The problem with such classifiers is, that they have to be trained in advance and only obstacles for which classifiers have been successfully created can be classified and tracked. Thus, this approach is not a general approach to avoid dynamic obstacles or use them for other navigation purposes. It is more an approach to detect and track specific obstacles.

3.1.1 Obstacle detection

Obstacle detection in general is often mentioned in the same context as self-driving cars, ground plane extraction or specifically road surface extraction as described in [67], [17], [57], [49].

In the literature, two main approaches to detect obstacles can be found. The first one is an extremely common one, especially in context with self-driving cars and traffic scenarios. It is about the analysis of the disparity map, especially the U disparity map and the V disparity map, given by a stereo vision system. All the approaches described in [67], [73], [57], [49] are generally based on this analysis and mostly just vary in pre and post filtering steps. The other approach is based on feature extraction in images and geometric approaches as for example described in [42]. Another completely different approach, which is a combination of ego motion subtraction and grid maps is presented in [52], [55]. All of them are described in detail below.

U-V disparity map based obstacle detection

As mentioned above a common method of extracting obstacles from the environment or background, is to analyze U-V disparity maps.

As it is described in [73], the U disparity map Δ_U as well as the V disparity map Δ_V can be calculated given a standard disparity map Δ_D delivered by a stereo vision system.

The U disparity map Δ_U has the size of $max_d \times image_w$, where max_d is the maximum disparity value and $image_w$ is the image width (number of columns).

A single column of Δ_U is representing the histogram calculated over the same column in Δ_D.

The V disparity map Δ_V can be calculated nearly the same way as the U disparity map. Δ_V has the size of $image_h \times max_d$, where $image_h$ is the image height (number of rows). A single row of Δ_V is representing the histogram calculated over the same row in Δ_D.

U and V disparity maps can be used for obstacle detection, since they have the following characteristics, as it is described in detail in [73].

- A projection of the horizontal ground to the V disparity map is represented by an diagonal line. Thus ground pixels belonging to a horizontal ground can be detected through detecting the diagonal line in the V disparity map.
- Projecting obstacles in the V disparity map will lead to a vertical line. The height of the obstacle can be gained through the length of this vertical line and the location of the vertical line represents the distance between the obstacle and the camera, the more left, the nearer. A vertical line intersecting a diagonal line, means that the obstacle is placed on the ground.
- The projection of an obstacle in the U disparity map is represented by a horizontal line, where the obstacle width can be calculated through the length of this horizontal line. The distance between an obstacle and the camera can be also calculated through the U disparity map. In this case the obstacle is near the camera, if the horizontal line is near the bottom of the U disparity map.
- Through the combined information of the U disparity map and the V disparity map the position and also the size of the obstacle can be calculated.

The line extraction itself is done by making use of any kind of edge filter, as for example the well known canny edge detector, and applying afterwards a line detection algorithm, as for example Hough line extraction. An alternative to the hough line detection is a so called region growing algorithm which is presented and used in [67].

As it is mentioned at the beginning of this section, obstacle extraction is mostly in the same context as self-driving cars and road surfaces extraction. Almost all the papers [67], [57], [49], found during the research definitely have a relation to traffic scenarios. Therefore, also the results presented within these publications are related to such traffic scenarios. That means, obstacles are mostly either cars or pedestrians, rarely seeded over streets or pedestrian zones. Unfortunately, there are no results presented about how this approaches are working in indoor scenarios or heavy crowded regions.

However, the work of [73] on the other hand is presenting an approach for obstacle detection in indoor environments, but in the results section it can be clearly seen, that this approach has problems with the detection of small or thin obstacles as for example table bases.

Small obstacle detection based on U-V disparity maps is especially tackled in [57]. But also this paper is strongly related to traffic scenarios. For testing

the presented approach, it seems like they are mainly using perfect rectangular shaped obstacles. This kind of obstacles do not really represent obstacles, which can be found in a complex environment, as for example round table bases or tubes in manufacturing buildings. Further, the general goal of [57] was to outperform a LIDAR obstacle detection algorithm and to detect obstacles smaller than a size of 15 cm and not to detect dynamic obstacles.

A slightly modified approach is presented in [67]. The focus of this article is especially on Advanced Driver Assistance Systems and on-road obstacle detection. The road detection itself is done by a fast color based road detection algorithm [68] and the obstacle detection is done with the help of U disparity maps. Also here it is hard to estimate how well this system will work for general ground detection and in indoor or heavy crowded environments since it was not tested in such situations.

After obstacles have been successfully detected, reference points of these obstacles are going to be passed to a kind of tracking algorithm or multi target tracking system, as for example a Kalman Filter.

Feature extraction based obstacle detection

A possible approach to solve the dynamic obstacle detection task based on feature detection, is shown in [42]. In this approach the scenery and all entire data delivered by the stereo vision system is going to be split into the two sections obstacles and environment. All the data in the section obstacles is further used for obstacle tracking and the data in the section environment can be used to make a reconstruction of the surroundings of the robot. Thus [42] is basically trying to detect all kind of obstacles around the robot and pass them on to a tracking algorithm.

The obstacle detection in general is based on a canny edge filter [12] followed by a Hough line extraction [6]. Canny edge filter is simply used, to extract all the pixels which are of interest for the following tracking process. Such pixels are for example edges from human contours, tables, doors, and so forth.

After the canny edge filter has been used, a Hough transformation is applied on the canny data, to search for long lines. Searching for such long lines is done, since edges corresponding with environmental structures have the characteristic of forming long lines. Thus, long lines are detected as environment and short lines are detected as obstacles.

Later on, the left image as well as the right image of the stereo vision system are used to obtain 3d data of the before extracted obstacles (3D reconstruction). After the 3d data was extracted, all the points belonging to the section obstacles are projected to an ground plane, also called XZ plane and passed on to the multi-target tracker XPFCP (extended Particle Filter with Clustering Process) presented in [42].

Finally, the information if the obstacle is dynamic or static can be figured out with the help of the data returned by the multi target tracker.

The drawback of the approach described in [42] is, that everything is completely based on the canny edge filtering and the Hough line extraction. There are a lot of parameters which has to be tuned quite well, to get a good performance of the Hough line detection [42]. Even if these parameters are tuned quite well, the whole system relies on the lines given by the Hough transformation

and it could be quite hard to distinguish between obstacles and environment. Obstacles which have a big rectangular shape for example, will be automatically sorted into the section environment and are not going to be passed to the multi target tracker, since the shape is containing long lines. On the other hand, in the most rare events the shape of a person can be represented by long lines. Thus, if the task is specifically about person detection and person tracking, this approach automatically reduces the wrong data passed to the multi target tracker. In the industry for example, a lot of trains, also called milk runs, are driving around and therefore there might be the possibility, that those milk runs are going to be detected as environment and not as an obstacle.

Occupancy Grid

Another completely different approach of detecting obstacles within a certain scene in presented in [52] and [55]. In this publications polar occupancy grids and a special grid cell merging algorithm are used to detect obstacles within the scene.

As mentioned in [52] occupancy grids a common method to represent the environment of the robot and where first introduced in [44]. When using occupancy grids especially in combination with stereo vision data many difficulties can occur. As described in [52], obstacles which are far away from the stereo vision sensor appear smaller and are therefore characterized by fewer points than obstacles which are near the stereo vision sensor, even if they have the same size in reality. To overcome this problems, in [52] a polar occupancy grid map with variable cell size is used. This ensures, that the point density is independent of the distance between the obstacle and the camera.

First of all, all 3D points are projected to an planar plane which is often also called XZ plane. Later on, all the points are associated to a certain cell which has the form of a trapezoid by the following formula:

$$\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} log_{1+k_r} \frac{Z}{Z_0} \\ f \cdot \frac{X}{Z \cdot k_c} \end{pmatrix} \tag{3.1}$$

where k_r is the compression factor along the z-axis (depth), k_c is the compression factor along the lateral x-axis, Z_0 is the distance between the origin of the sensor frame and the first row of the grid and f is representing the focal length. Z and X are representing the depth and the lateral coordinates of the point.

After all points have been associated with a cell, cells which are belonging to the same obstacle are merged. This merging approach is based on the distance between the cells and also on additional information as for example speed and direction of motion. This additional information is gathered from a simple ego motion compensation. Thus, the difference of optical flow and visual odometry.

The advantage of this approach is, that it is able to run a rate of 25 frames per second on a simple core 2 Duo machine and the final found obstacle can be easily used within further methods as for example for tracking purposes [52].

3.1.2 Dynamic pixel detection

As mentioned at the beginning of this section, dynamic pixel detection based on optical flow algorithms is another possibility to detect dynamic obstacles. In general, optical flow algorithms are mostly used in the same context with dynamic obstacle detection, but they are also used for other fields within robotics as for example for visual odometry calculation as shown in [41].

A very straight forward and new approach to detect dynamic pixels is presented in [69] and [75]. Where the basic idea of [69] is equal to the one described in [75] and the articles in general differ only a little bit. The main difference will be discussed in detail in the following. But at first there will be a short overview of the general structure of the approach.

The approaches in [69] as well as in [75] start with the computation of the optical flow. Beside the calculated optical flow also the disparity map, the odometry and confidence values will be further inputs to these approaches. In this research we are mainly focusing on the basic idea and will not go into details of visual odometry calculation or confidence value calculation, since there is the assumption that this data is directly retrieved from the stereo vision system. Anyways, a detailed description can be found in both articles, [69] as well as [75].

After the optical flow calculation, a motion likelihood estimation for every single pixel is done. At last step a min-cut max-flow segmentation algorithm to separate foreground obstacles from background obstacles is applied.

Optical flow

In the past few years the topic optical flow is under heavy investigation and since stereo vision systems and also mono camera systems are getting used to be a standard sensor for robots of any kind. This topic is getting more and more important for the entire robotics and vision community. There are even competitions and benchmark suits, as for example the KITTY Benchmark Suit, which are providing data sets to compare different optical flow algorithms with each other [23]. Therefore, in the following section there will be a pretty rough overview of different available methods.

Basically, optical flow algorithms can be divided into the two sections sparse algorithms, as for example the well known Lucas-Kanade method [5] and dense algorithms as presented by Horn and Schunck in [25] or the classical Farneback algorithm presented in [19]. Dense optical flow computation means, that the optical flow is calculated for every single pixel. Sparse on the other hand means, that the optical flow is only calculated for specific points or regions of interest. Thus the benefit of dense algorithms lies within the fact, that there is a maximum coverage of the scene, with the drawback, that the needed time to compute the optical flow is higher than with sparse algorithms [13]. Criteria to compare optical flow algorithms are besides time complexity for example average error in pixels and density (coverage) [7].

The basic Lucas-Kanade algorithm was already introduced in 1981 as an image registration algorithm [38] and has become one of the most widely used

techniques in context with optical flow, tracking, layered motion, mosaic construction and face coding [5]. On the other hand there is the well known method for dense optical flow computation presented by Horn and Schunk [25].

Nowadays these classical algorithms are rarely used in its original implementation, they are more used as a foundation and almost all new methods are based on these as for example eFOLKI presented in [54].

Since Neural Networks have a very wide range of application and are nowadays used for almost all applications, it is not surprising, that most new approaches handed in and tested at the KITTY Benchmark Suit are based on this technique.

One of the most popular Neural Networks which is used for optical flow computation at the point in time this research has been done, is the so called FlowNet presented in [14]. This Neural Network is based on a convolutional architecture and reached the 97 place out of 109 at the KITTY benchmark suit based on the parameter Out-Noc, which means percentage of erroneous pixels in non-occluded areas, which was at this point of time 37.05 percent. Other well known approaches based on Neural Networks which can be found at the KITTY Benchmark Suit are for example FlowFieldCNN [3] or DeepFlow [71].

At the moment, the focus of most recent works in optical flow extraction is on accuracy and not on time complexity, but time complexity is especially crucial for real time use [31]. Therefore, the method proposed in [31], called Dense Inverse Search is focusing on a good trade off between time complexity and accuracy for a dense optical flow calculation.

The presented method is mainly based on the efficient search of correspondences described in [5] and the inverse compositional image alignment proposed in [4]. As it was shown in [31], based on the average end point error per pixel, the Dense Inverse Search method is in any case outperforming the Farneback method. At a runtime of 10 Hz this approach has an error of about 6 pixels, which is nearly the same error as for SparseFlow and DeepFlow, where these networks are clearly slower then the Dense Inverse Search approach. On the other hand, by increasing the time horizon, the smallest error Dense Inverse Search can reach, is about 5.5 pixels, where FlowNet and SparseFlow are able to compute the optical flow with an error clearly lower than 5 pixels.

Pixel motion likelihood

According to the pipeline presented in [75], the next step after the optical flow calculation which has to be done is to compute the pixel motion likelihood. At this point, there has to be mentioned that the pixel motion likelihood calculation presented in [69] and [75] depends on dense optical flow data as input, since the goal is to calculate the motion likelihood for every single pixel.

As it is described in [75], to get the motion likelihood for all pixels, besides the **optical flow**, the **global image motion flow** and following from this the **residual image motion flow** has to be calculated as well.

The global image motion flow is in principle based on the prediction of future pixel positions and is needed to get the motion flow which is only caused by the

camera motion. Thus, motion which is produced by moving the robot in any arbitrary direction.

The new position $p_t = (u, v, 1)^T$ of a pixel at the image plane can be predicted based on the position $p_{t-1} = (u, v, 1)^T$ of the previous image frame and the following Equation 3.2:

$$p_t = K \cdot R \cdot K^{-1} \cdot p_{t-1} + \frac{K \cdot tr}{Z_{t-1}} \tag{3.2}$$

, where the camera intrinsic parameter matrix K is shown in Equation 3.3:

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

and K^{-1} is the inverse of K. f_x and f_y are representing the focal length in pixels, s is the skew and x_0 respectively y_0 is the position of the principle point [56].

The overall relative camera rotation is represented by R and tr is representing the relative camera translation between two consecutive time stamps and Z_{t-1} represents the depth of a 3D point at time stamp $t-1$.

This prediction is only valid for 3D points which are coming from static obstacles, and it does not hold for dynamic obstacles.

The global image motion flow g for the whole image can be calculated by 3.4:

$$g = \begin{pmatrix} u_t - u_{t-1} \\ v_t - v_{t-1} \end{pmatrix} \tag{3.4}$$

, where g is:

$$g = \begin{pmatrix} g_u \\ g_v \end{pmatrix} \tag{3.5}$$

Finally, the **residual image motion flow** q can be calculated by using Equation 3.6, which is combining the values returned by the **global image motion flow** and the measured **optical flow**:

$$q = g - m = \begin{pmatrix} g_u - m_u \\ g_v - m_v \end{pmatrix} \tag{3.6}$$

The optical flow, delivered by any dense optical flow algorithm mentioned above, for a certain point with position (u,v) in the image plane is represented by:

$$m = \begin{pmatrix} m_u \\ m_v \end{pmatrix} \tag{3.7}$$

In an ideal world the residual image motion flow should be zero for static points and bigger than zero for dynamic points. However, as it is mentioned in [75] this is not true, because of existing uncertainties. In [75] there are listed four possible types of uncertainties:

1. error in **camera motion estimation**
2. error in **depth estimation**
3. error in **optical flow estimation**
4. **image noise** (image rectification, camera intrinsic parameters, digital image quantization)

, where the uncertainty of the measured optical flow is not considered, since it only affects the detection results locally. As it is mentioned in [75], all uncertainties except the resulting from the optical flow, should not be ignored, since this could lead to a large number of false positive detections.

Finally, after the residual image motion flow has been computed, the motion likelihood for every single pixel, considering the above listed uncertainties, can be calculated.

As it is described in [75], to handle the problem with the uncertainties, the errors are propagated from the sensor to the final estimation using a first order Gaussian approximation. The residual image motion flow covariance matrix can be calculated by using the following Equation 3.8.

$$\Sigma_{RIMF} = J \cdot \Sigma \cdot J^T \tag{3.8}$$

In Equation 3.8 J is representing the Jacobian matrix of Equation 3.6 with respect to the input variables (camera motion Θ, pixel position (u,v) and the disparity value d). Note, that as mentioned above the uncertainty of the optical flow is not considered here. $\Sigma = diag(\Sigma_\Theta, \Sigma_0)$ is the covariance matrix of all input variables, where Σ_Θ is the covariance matrix of the camera motion and Σ_0 is the covariance matrix of the disparity estimation process. Σ_0 is based on the variances of the pixel quantization errors Σ_u, Σ_v and Σ_d. As mentioned in [75], the uncertainty of the disparity value Σ_d can be linearly approximated by:

$$\sigma_d(u, v) = \sigma_0 + \gamma \cdot U_d(u, v) \tag{3.9}$$

, where σ_0 and γ are two constant parameters and $U_d(u, v)$ is the uncertainty of the disparity value at position (u,v).

Finally, the motion likelihood for a single pixel can be retrieved by making use of the residual image motion flow q and Σ_{RIMF}:

$$\mu_q = \sqrt{q^T \cdot \Sigma_{RIMF}^{-1} \cdot q} \tag{3.10}$$

Note, that 3.10 is representing the Mahalanobis distance [40].

Min-cut max-flow segmentation

After the motion likelihood for each single pixel has been calculated, all pixels can be categorized either into to the category static or dynamic. This can be represented by the following binary labelling [69]:

$$L(x) = \begin{cases} 1 & \text{if the pixel x is part of a dynamic obstacle} \\ 0 & \text{if the pixel x is part of a static obstacle} \end{cases} \tag{3.11}$$

A first attempt would be to apply a predefined threshold on the motion likelihoods. As it is shown in [75], the results after applying a fixed static threshold are not satisfying, since an optimal threshold cannot be found. This is due to the imperfect measured optical flow.

To get an optimal assignment of pixel to categories, Equation 3.11, a segmentation algorithm has to be applied, where the following constraints have to be considered [75]:

1. pixels with high **motion likelihood** should be detected as moving
2. adjacent pixels with similar **appearance** should share the same label
3. adjacent pixels with similar **distance** should share the same label

All the above listed constraints are combined in the following energy function [75]:

$$E(L) = E_r(L) + \lambda E_b(L) \tag{3.12}$$

, where E_r is the so called region term and E_b is the so called boundary term. λ in this case is used to balance the influence of the boundary term E_b.

The motion likelihood of each single pixel is considered in the region term E_r by Equation 3.13:

$$E_r = -\sum_{x \in \Omega} L(x) \cdot \xi_m(x) + (1 - L(x)) \cdot \xi_s(x) \tag{3.13}$$

, where Ω represent the image domain, $\xi_m(x)$ is the motion likelihood of a single pixel given by Equation 3.10 and $\xi_s(x)$ is the belief that the pixel is static. $\xi_s(x)$ is selected in advance as a static value.

The appearance and distance constraint listed as second and third constraint in 3.1.2 are both considered within Equation 3.14:

$$E_b = \sum_{x \in \Omega} \sum_{\hat{x} \in N_4(x)} (B^d(\hat{x}, x) + B^c(\hat{x}, x)) \cdot |L(\hat{x}) - Lx| \tag{3.14}$$

Thus the boundary term E_b is used to favour labellings of neighboring pixels to be identical [69]. $B^d(\cdot)$ and $B^c(\cdot)$ can be calculated by Equation 3.15 and N_4 is capturing the 4 neighborhood pixels (upper, lower, left, right).

At this point of the research, there has to be mentioned, that there is a difference between [75] and [69]. While [69] only considers the appearance information, in [75] both, appearance as well as depth information are considered in the boundary term E_b. In [75] the depth constraint is included, since the depth of moving obstacles usually has a big difference with the lateral background [11], [61].

As described in [75], B^d and B^c can by calculated by the equations 3.15:

$$\begin{aligned} B^d(x_i, x_j) &= exp(-\sigma \cdot (|Z(x_i) - Z(x_j)|) + \alpha) \\ B^c(x_i, x_j) &= exp(-\sigma \cdot (|I(x_i) - I(x_j)|) + \alpha) \end{aligned} \tag{3.15}$$

, where $Z(x)$ is the disparity value for pixel x and $I(x)$ is the appearance/gray scale of the pixel x. σ and α are constants to control the descent speed and peak value of the positive, monotonically decreasing function $B(\cdot)$. In [75] σ and α where set to $\sigma = \sqrt{2}$, respectively $\alpha = 0$

Finally, the resulting energy function can be written as:

$$\begin{aligned} E = &\sum_{x \in \Omega} \{-L(x) \cdot \xi_m(x) - (1 - L(x)) \cdot \xi_s(x) \\ &+ \sum_{\hat{x} \in N_4(x)} (B^d(\hat{x}, x) + B^c(\hat{x}, x)) \cdot |L(\hat{x}) - Lx|\} \end{aligned} \tag{3.16}$$

As described in [69], finding the minimum of the energy function shown in Equation 3.16 is the same as finding the s-t-separating cut with minimum costs of a particular graph $G(v; s; t; e)$. The graph G consist of node $v(x)$ for every single pixel, as well as a source node labelled as s and a target node labelled t. Edges are added by connecting each node n with the source node s and the target node t as well by connecting each node n with its four neighboring nodes. The costs for individual edges can be taken from Table 3.1 shown below.

edge	**edge costs**
source link: s $\rightarrow v(x)$	$-\xi_m(x)$
target link: $v(x) \rightarrow t$	$-\xi_s(x)$
N_4 neighborhood: $v(\hat{x}) \leftrightarrow v(x)$	$B^d(\hat{x}, x) + B^c(\hat{x}, x)$

Table 3.1: Min-cut max-flow edge costs

The s-t cut with minimum costs can be finally computed by using min-cut/max-flow algorithm as presented in [11] and [61]. After the algorithm has been applied, all nodes which are connected to the source node, can be labelled as static pixels and on the other hand, all nodes which are connected to the target node can be labelled as dynamic.

In [75], it is further stated, that λ, which is used to balance the influence between the region term E_r and the boundary term E_b has an essential influence to the final segmentation results. If a low value for λ is chosen, the segmentation is mainly based on the motion likelihood of a single pixel, where a high value of λ results in only small or no segments at all [75].

If computational time should be reduced, a down sampling method for the image can be used as also mentioned in [75].

As it is stated in [75], the computational time for a single image is about 165 seconds based on a Matlab implementation. Thus even if this new approach is creating really good and accurate results it can not be used for real time applications.

3.2 Dynamic obstacle tracking

In this section we are focusing on dynamic obstacle tracking. Below, classical approaches as for example the Kalman Filter, as well as new approaches like tracking with Neural Networks are mentioned. Obstacle tracking and especially the related topic data association is again a pretty large sector. Thus also in this section of the research we try to focus on the basic principles and the main ideas of the different methods.

The purpose of dynamic obstacle tracking for this thesis is to get motion information from different moving obstacles. Motion information captures for example velocity, acceleration, as well as velocity of direction change and the direction of motion itself. The input for the different tracking algorithms can be for example the current position of an obstacle, but also a bounding box in an image containing the obstacle. The position or the bounding box of an obstacles can be gathered by a sensor or by any other system, where often the output of the obstacle detection algorithm is directly used as input for the tracking algorithm. In the literature the input for tracking algorithms is mostly called observation.

All the different tracking algorithms are based on two steps, which are prediction and update, or prediction and correction. Therefore, at each time stamp t, it is tried to predict the future position for time stamp $t + 1$ of an obstacle. After the prediction has been done, it is tried to reduce the uncertainty for future predictions, by correcting the current prediction with a related observation. Exactly at this point a further difficulty is coming up. Associating predictions with observations is not an easy task and there can easily occur miss associations which lead to bad tracking behavior and thus to wrong motion information.

As it is reported in [34], till the year 2015 preferable strong global optimum methods were used to solve the association problem. Linking the observations with already existing tracks was often cast with a graphical model and solved as for example with k-shortest paths or simplex algorithm [34].

Nowadays, the top performing methods for obstacle tracking applied on images, are taking into account also the appearance of an obstacle. Such top performing methods are using for example sparse appearance models [18], online appearance update [29] or integral channel feature appearance models [28].

As well as for optical flow algorithm comparison, also for tracking algorithm comparison there is a benchmark available which is called Multiple Object Tracking benchmark, or short MOT. Within MOT, there are two different challenges MOT15 as well as MOT16 available, where the later one is presented in detail in [34]. According to [34], nowadays a huge problem with the different published tracking methods is, that a lot of these methods are fine tuned to individual sequences. This means, that these tracking algorithms are performing pretty well on selected individual scenes, but do not perform well on sequences with a high variety. To overcome this problem, in [34] a new benchmark is introduced which is exactly dealing with this difficulty and enables the comparison of tracking algorithms based on real life scenarios.

Since only real time tracking algorithms are of interest for this thesis, the remaining discussion is only focusing on real time tracking algorithms. Based

on the ranking presented in [34], the top three of the best real time tracking algorithms are LP2D [33], DP_NMS [53] and JPDA_m [60]. These three methods are compared in Table 3.2, based on the criteria multi object tracking accuracy (MOTA) and multi object tracking precision (MOTP) of the challenge MOT16.

Rank	**Name**	**MOTA** [%]	**MOTP**[%]	**Association Method**
1	LP2D [33]	35.7	75.8	simplex algorithm
2	DP_NMS [53]	32.2	76.4	k-shortest paths
3	JPDA_m [60]	26.2	76.3	joint probabilistic DA

Table 3.2: Comparison of top 3 real time trackers listed in [34] based on MOT16

As described in [34], accuracy is capturing missed targets, ghost tracks and identity switches, while precision is measuring how well the objects are localized.

In [33], the tracking problem and the data association problem is converted to a minimum-cost network flow problem, and finally solved by using Linear Programming (Simplex Algorithm). Besides the independent motion of individual obstacles, also the complex interaction between moving obstacles, as for example pedestrians, is considered [33].

In [53], in general a greedy algorithm that sequentially instantiates tracks, by using shortest path computations on a flow network is presented. The benefit of such a greedy approach lies in the fact, that it is easy, simple, scalable and allow it to make use of pre processing steps within the tracking algorithm [53]. The shortest paths itself is calculated by the dynamic programming algorithm presented in [53].

In [60] an algorithm based on joint probabilistic data association is proposed, which tries to find m-best solutions to an integer linear program.

All the three methods shortly described above are used for pedestrian tracking. Pedestrian tracking in general is not an easy task and there are a number of difficulties which has to be considered. Such difficulties are for example group motions, occlusion of pedestrians by other pedestrians or fast change of direction and speed.

Not only for optical flow computation but also for tracking, Neural Networks especially Recurrent Neural Networks are becoming more and more popular. Until the year 2017 only little work related to deep learning and multi-target tracking has been done [45]. The reason for this is, that deep models require a huge amount of training data which is not available yet and further a large number of parameters have to be tuned [45]. In [45] an end to end Recurrent Neural Network which can handle all relevant multi target tasks as for example prediction, data association, track initiation and termination is presented. The final method was tested on the MOT15 challenge and reached a MOTA of 19 percent and MOTP of 71 percent. To make this method comparable to the others mentioned above, the results of all algorithms based on MOT15 are listed in Table 3.3, where the metric is the same as above.

A pretty simple and lean approach, which is definitely also bringing state of the art results is called Simple Online And Realtime Tracking, or short SORT presented in [9]. This approach reaches based on MOT15 an average MOTA of

Rank	Name	MOTA [%]	MOTP [%]
1	SORT [9]	33.4	72.1
2	JPDA_m [60]	23.8	68.2
3	LP2D [33]	19.8	71.2
4	RNN_LSTM [45]	19	71
5	DP_NMS [53]	14.5	70.8

Table 3.3: Ranking of mentioned tracking algorithms based on MOT15.

33.4 and an average MOTP of 72.1, which means that the approach presented in [9] is absolutely competitive to all approaches listed in Table 3.3. The benefit of the approach shown in [9], lies in its simplicity. For the tracking/optimization a standard Kalman Filter based on a constant velocity model is used. The data association problem is solved by the well known hungarian algorithm originally introduced in [32] based on the statistical distance given by the Mahalanobis equation [40].

3.3 Navigation in Crowded Dynamic Environments

In the following section, methods used for enabling navigation in crowded dynamic environments and especially dynamic obstacle avoidance are described. Note that only approaches which are related to wheel based or car like robots are captured within this section. Path planning itself is not only used within mobile robots, its also used for applications like circuit board designs, network routings, computer animations, pharmaceutical drug design and so on [59].

As mentioned in [51] navigation in dynamic environments and obstacle avoidance, especially dynamic obstacle avoidance, are one of the fundamental problems of mobile robots. The goal of robot navigation is to find and execute a collision free path from the current position of the robot to a given goal, while optimizing a performance criterion such as distance, time, or energy. Mostly distance is the criteria which should be optimized [59].

To tackle this, two navigation categories are available: global navigation and local navigation, in the literature also mentioned as off-line navigation and on-line navigation.

For the global navigation, prior knowledge of the environment is used for the path planning [51]. Thus, stationary points of static obstacles are considered within this phase of the path planning. As described in [51], for the global path planning grid [70] or graph representations are commonly used, where the global path itself can be calculated as for example with the help of the Dijkstra algorithm [63] or the well known A* Algorithm originally presented in [24].

As mentioned in [59], local path planning begins its initial path with the path given by the global planner, but switches to the local mode when it discovers changes in the environment as e.g. dynamic obstacles.

Thus, roughly summarized global path planning is dealing with finding an optimal path to a certain goal while local path planning is dealing with avoiding dynamic obstacles which are not considered by the global planner.

In the following sections only local path planning algorithms are described, since the goal of the presented work is to avoid dynamic obstacles.

All the algorithms which can be used for local path planing can be categorized into the sections classical or evolutionary approaches [59].

3.3.1 Classical Approaches

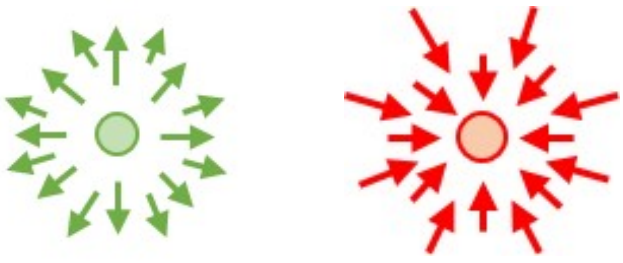
In [59] are described different classical approaches such as for example vector field histograms [10], dynamic window approach [21], but also potential fields [27] and collision cones (velocity obstacles) [20] are described.

As described in [59] for the vector field histograms [10] approach, at every instant a polar histogram is generated to represent the density of all the obstacles around the robot. After this polar histogram has been created, the robot steering direction is chosen according to the polar histogram section with the least density. As explicitly mentioned in [59], this method is more suited for environments with sparse moving obstacles.

The dynamic window approach [21] on the other hand considers all feasible linear and angular velocities. Based on the acceleration capabilities of the robot, a optimized velocity and direction for the next instant is calculated [59].

Potential Fields Approach

The potential fields approach which was originally introduced in 1986 by Khatib [27], as it is written in [59], is a very popular and nowadays a pretty common approach to do local navigation. By this approach, a robot is treated as point robot moving under the influence of an Artificial Potential Field. Such a field can be represented by an array or field of vectors, where each vector $v = (m, d)^T$ represents a force and consists of a magnitude m and a direction component d. An Artificial Potential Field is basically a combination of attractive forces generated by goals and repulsive forces generated by obstacles, as shown in Figure 3.1.



(a) Attractive forces. (b) Repulsive forces.

Figure 3.1: Potential field forces.

As presented in [27], an Artificial Potential Field can be generated with the help of the so called potential field function $U(q)$ presented in Equation

3.17, where q is the xy point position of the robot, $U_{att}(q)$ is representing the attractive field and $U_{rep}(q)$ the repulsing field.

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{3.17}$$

The function $U(q)$ must be differentiable, since the final robot velocity can be set proportional to the force $F(q) = - \bigtriangledown U(q)$ generated by a potential field.

Thus, the Artificial Force Field can be retrieved by:

$$F(q) = - \bigtriangledown U(q) \tag{3.18}$$

$$F(q) = F_{att}(q) + F_{rep}(q) \tag{3.19}$$

$$F(q) = - \bigtriangledown F_{att}(q) - \bigtriangledown F_{rep}(q) \tag{3.20}$$

The Attractive Potential Field $U_{att}(q)$ is often written as the parabolic function or a linear function representing the Euclidean distance to the goal:

$$U_{att}(q) = \frac{1}{2} \cdot k_{att} \cdot p_{goal}^2(q) \tag{3.21}$$

, where k_{att} is a positive scaling factor, and $p_{goal}(q)$ is the Euclidean distance between the robot position q and the goal position q_{goal}.

The attractive force $F_{att}(q)$ can be retrieved by:

$$F_{att}(q) = - \bigtriangledown U_{att}(q) \tag{3.22}$$

$$F_{att}(q) = -k_{att} \cdot (q - q_{goal}) \tag{3.23}$$

The Repulsive Potential Field $U_{rep}(q)$ is used to create a kind of barrier around all the obstacles. This field should be strong close to the obstacle and have no influence far from the obstacle. Therefore, Repulsive Potential Field $U_{rep}(q)$ can be represented by:

$$U_{rep}(q) = \begin{cases} \frac{1}{2} \cdot k_{rep} \left(\frac{1}{p(q)} - \frac{1}{p_0} \right)^2 & \text{, if } p(q) \leq p_0 \\ 0 & \text{, otherwise} \end{cases} \tag{3.24}$$

, where p_0 is the distance of influence and p_q is the minimal distance to the obstacle.

Again, the derivative of the repulsive potential field $U_{rep}(q)$ can be calculated to obtain the resulting force $F_{rep}(q)$:

$$F_{rep}(q) = - \bigtriangledown U_{rep}(q) \tag{3.25}$$

$$F_{rep}(q) = k_{rep} \cdot \left(\frac{1}{p(q)} - \frac{1}{p_0} \right) \cdot \frac{1}{p^2(q)} \cdot \frac{q - q_{obstacle}}{p_q} \tag{3.26}$$

The finally generated robot movement is similar to a ball rolling down a hill. As described in [59], this algorithm is very popular and known because of its simplicity and mathematical elegance. Nevertheless, there is also a big drawback. There might be the possibility, that the robot become stuck if there are equal magnitudes of attractive and repulsive forces. Thus, the robot is trapped in a local minimum and cannot move anymore. This scenario can be overcome with an extension to this algorithm which is called escape-force algorithm.

A special approach including dynamic obstacles in potential fields is presented in [22], where a new potential field function and further virtual forces are introduced.

Velocity Obstacles

Velocity Obstacles where introduced originally in 1998 [20] and are a commonly used concept to avoid static as well as dynamic obstacles in the velocity space. The avoidance itself, is only based on the current position and velocity of the robot and the obstacles [20].

As it is assumed in [20], also for the further descriptions in this section the following explanations and formulas are restricted to circular, non rotating robots and obstacles for simplicity reasons. In general, this is not a limitation, since polygons can be represented by a number of circles [50].

As described above, a robot and an obstacle can be represented by two circles labelled A and B, where at time t_0 both are moving with linear velocities v_A and v_B. To compute a velocity obstacle, first B has to be mapped into the configuration space of A. This is done by reducing circle A to point $\hat{A}$ and enlarging circle B by the radius of A to $\hat{B}$ [20]. Further, a potential collision cone of a velocity obstacle can be define as:

$$CC_{A,B} = \{v_{A,B} | \lambda_{A,B} \cap \hat{B} \neq 0\} \tag{3.27}$$

, where $v_{A,B}$ is defined as:

$$v_{A,B} = v_A - v_B \tag{3.28}$$

and $\lambda_{A,B}$ is the line of $v_{A,B}$ [20].

Based on the descriptions in [20], the above defined collision cone $CC_{A,B}$ can be represented by a planar sector with its origin in $\hat{A}$, bounded by the two tangent λ_f and λ_r. All the velocities between the two tangents λ_f and λ_r will cause by definition a collision between the robot and the obstacle, if both of them keep their current velocity, direction and shape. Such a collision scenario is illustrated at the Figure 3.3 below.

If multiple obstacles should be considered there is commonly used an absolute velocity representation to simplify calculations. This can be done by adding the velocity v_B to each velocity within the cone $CC_{A,B}$, or by just translating $CC_{A,B}$ by v_B [20] as it is illustrated at the Figure 3.2 below.

Finally the Velocity Obstacle can be define as:

$$VO = CC_{A,B} \oplus v_B \tag{3.29}$$

, where $\oplus$ is representing the Minkowsky vector sum operator.

As it is described in [20], multiple obstacles can be considered by using the union of the individual velocity obstacles:

$$VO = \bigcup_{i=1}^{m} VO_{B_i} \tag{3.30}$$

, where m is representing the number of obstacles. In general, if there are multiple obstacles, they will get prioritized based on some criteria as for example remaining time till collision.

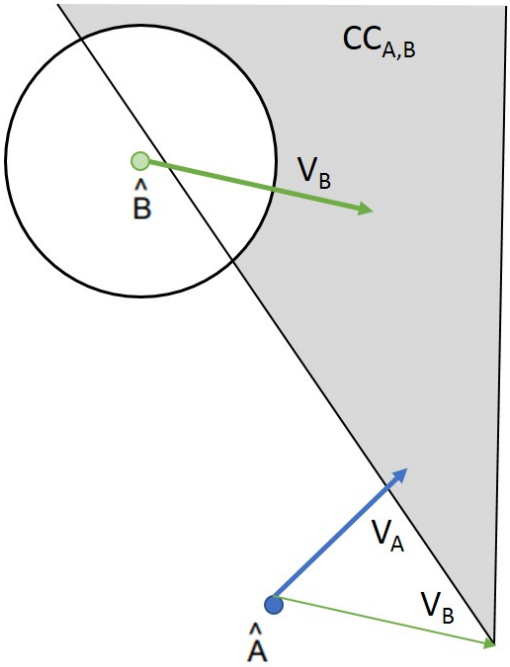


Figure 3.2: Velocity obstacle based on absolute velocities.

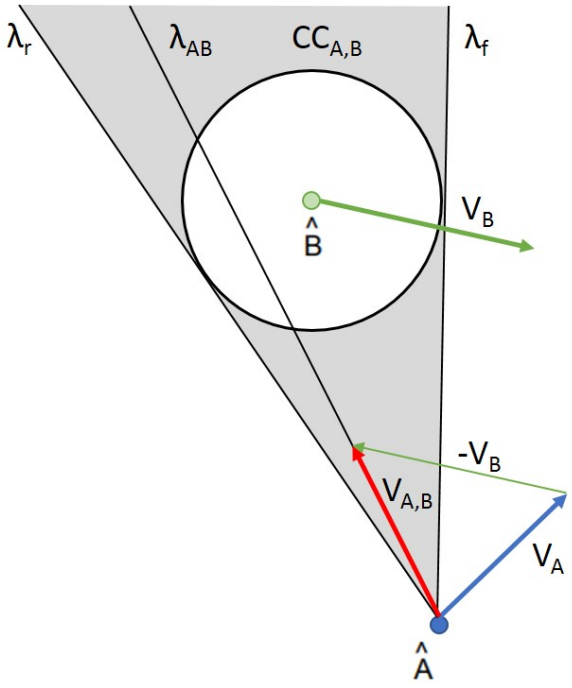


Figure 3.3: Collision cone based on relative velocities.

3.3.2 Evolutionary Approaches

Evolutionary approaches in general are nowadays often used, to overcome problems arising in the classical approaches as described above. Further, due to NP-hard complexity of path planning problems, classical approaches are also often combined with evolutionary approaches to save computational time and keep the update rate high [59].

As described in [59], to overcome the drawback of getting stuck in a local minimum as described in the section Potential Fields above, the described procedure can be combined with a genetic algorithm [1]. In [66], as for example, an algorithm called escape-force algorithm is introduced. It can be used to retrieve an optimal potential field function based on genetic algorithms to recover the robot from the trap.

In [39], a reactive immune network, to overcome the local minima problem is introduced. The potential field approach is coupled with biologically inspired immune network and the overall response of the immune network is calculated using genetic algorithms [59]. A general drawback of genetic algorithm is, that these algorithms are non-deterministic. This simply means that results can vary even if exactly the same input is given [51].

At this point there has to be mentioned, that evolutionary approaches are only described within this research for completeness. Since the goal of this work is to include dynamic obstacle avoidance in the local planner of an already existing navigation stack integrated in already operating robots, implementing a completely new local planner based on evolutionary approaches was not an option.

Chapter 4

Concept

In this chapter of the thesis the concept for solving the given problem is presented. Thus, all steps, algorithms and methods which are needed to tackle the goal of this thesis are explained. As a recap, the goal of the thesis is to enable dynamic navigation in crowded environments.

First of all, a general concept overview is presented. This overview is followed by detailed descriptions focusing data preprocessing, obstacle detection, obstacle tracking, and navigation.

4.1 Concept Overview

This section of the chapter is used to provide a general overview of the concept.

As input data a 3D point cloud delivered by a stereo vision system or a 3D laser range scanner is assumed. As mentioned in the chapters 5 and 6, the resolution of the camera, has a direct influence to the overall outcome, since the maximum depth is directly related to the resolution. At first this data has to be preprocessed, since the delivered data is raw data. Thus, data which includes noise and also wrong measurements. Preprocessing in this context simply means, to apply filters which can reduce this noise and get rid of not needed or unnecessary points. At next the preprocessed data is passed to an obstacle detection module, followed by a tracking module. Finally, the results of the tracker are given to the navigation stack, where the best trajectory is selected as the new local path and is further executed.

The concept overview is illustrated at Figure 4.1. All the modules and used algorithms are described in detail in the following few sections.

4.2 Dynamic Obstacle Detection

In this section, all methods and algorithms needed to extract dynamic obstacles from 3D data are described.

First of all, a preprocessing step has to be done. This step is followed by an obstacle detection and obstacle tracking step. In general, a new iteration of the entire process is triggered by new data delivered by the sensor. The final output of this module is a set of detected dynamic obstacles and the information about position, speed and direction of the individual obstacles.

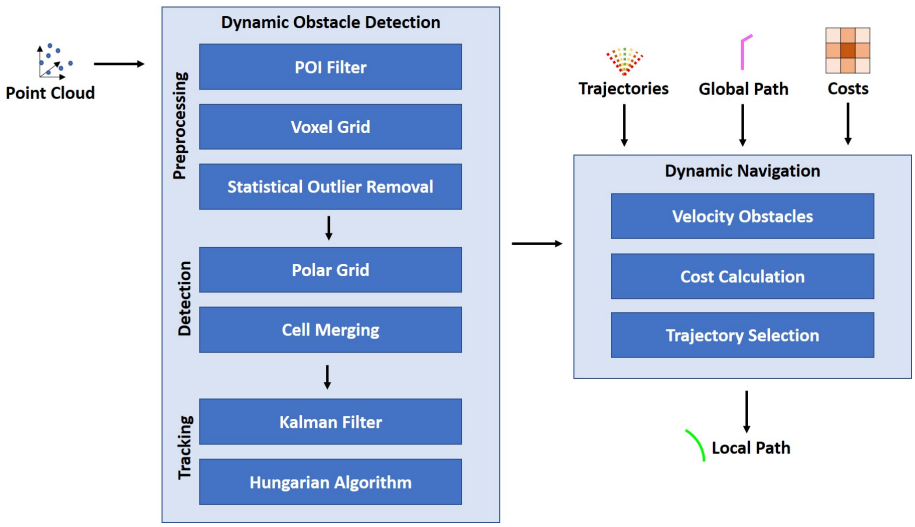


Figure 4.1: Concept overview.

4.2.1 Preprocessing

As mentioned above, the data delivered by the sensor has the form of a 3D point cloud. In general, this data contains noise. This means, that not all coordinates of the points are correct and correspond exactly to points in the real environment. Based on the sensor type this noise can be higher or lower. In our case, two main noise sources have to be distinguished. Measurement errors, caused by sensor uncertainty and for stereo vision systems especially, reconstruction errors. Reconstruction errors can occur, due to missing texture or overexposure and it follows that is not always possible to find the correct related pixels for the triangulation described in Section 2.2.1. To reduce the noise in the data, the statistical outlier removal algorithm, detailed explained in Section 2.3.3, is applied to the point cloud. Since the runtime of this algorithm is proportional to the number of points, it is tried to minimize the number of points in the point cloud before applying the algorithm.

To minimize the number of points, a area of interest is defined and all points which are not within this area, are removed from the point cloud. Since points which are belonging to the ground are not representing an obstacle, all points which are lower than a certain height above the ground are also removed, where the ground is assumed to be a flat surface. The algorithm which is used to extract these points of interest is shown in Algorithm 1 , where min_x, min_y, min_z as well as max_x, max_y, max_z are predefined minimum and maximum thresholds for all axes of the coordinate system.

After all points of interest haven been extracted, the density of the point cloud is reduced by applying a voxel grid, as described in detail in Section 2.3.2. Finally, the filtered point cloud is given to the detection module.

The overall preprocessing pipeline is shown in Figure 4.2.

Algorithm 1 POI extraction

Input: raw sensor point cloud P

Output: new point cloud Q containing all points of interest

```
for point p in P do
  if min_x ≤ p.x ≤ max_x and
  min_y ≤ p.y ≤ max_y and
  min_z ≤ p.z ≤ max_z then
    Q.add(p)
  end if
end for
```

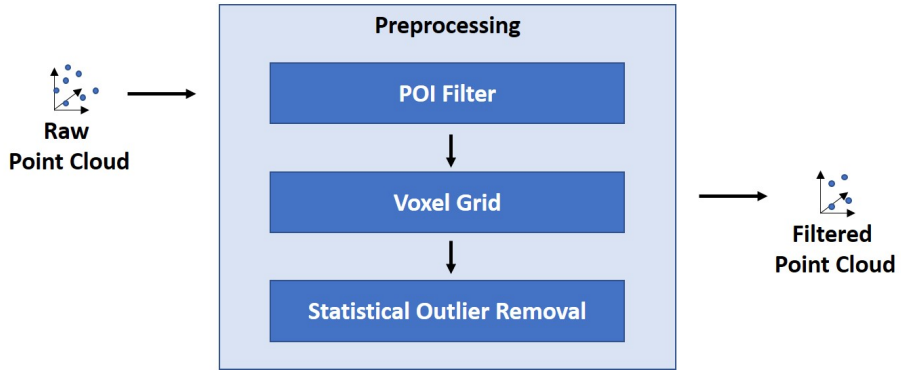


Figure 4.2: Preprocessing pipeline.

4.2.2 Detection

Obstacle detection is mainly based on two steps. First of all, a polar grid has to be created and all cells of this grid, which are belonging to the same obstacle are merged.

The polar grid creation is based on the method presented in [52]. We use Equation 4.1:

$$\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} log_{1+k_r}(\frac{Z}{Z_0}) \\ f \cdot \frac{X}{Z \cdot k_c} \end{pmatrix} \tag{4.1}$$

, detailed explained in Section 3.1.1 For each point in the filtered point cloud given by the preprocessing pipeline, a corresponding cell is determined. Such a cell can be seen as a container storing all the points which are belonging to it.

A polar grid is used instead of a normal grid, to overcome the problem of point density and obstacle appearance mentioned in Section 3.1.1. Each cell of this polar grid can be represented as a trapezoidal and due to Equation 4.1, the size of a single trapezoidal is based on the distance to the camera.

To reduce noise and wrong detections, all cells which are containing less points than a certain predefine threshold are emptied and not considered in the later merging process. The polar grid creation procedure is presented below in Algorithm 2 , where the function $calculateKey(\cdot)$ calculates the values i and j of Equation 4.1.

Algorithm 2 Polar grid creation

Input: preprocessed point cloud P
Output: polar grid G

```
G ← new PolarGrid()
for point p in P do
  key ← calculateKey(p)
  cell ← G.getCell(key)
  cell.addPoint(p)
end for
```

After all points have been assigned to their corresponding cells, a cell merging algorithm is applied to the polar grid. The goal of this merging algorithm is to find cells which are belonging to each other and finally to separate different obstacles. A pseudo code and the description of the cell merging algorithm is presented in Algorithm 3.

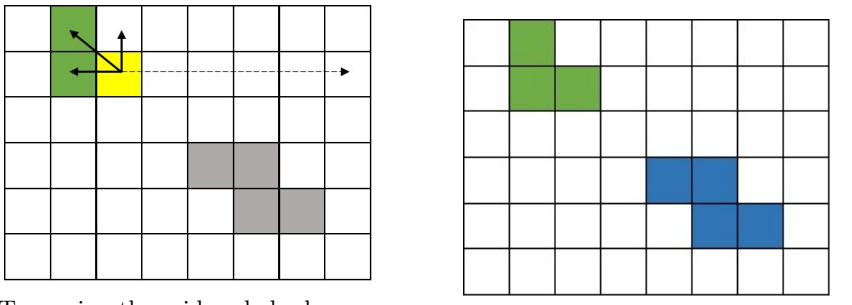
Algorithm 3 Cell Merging

Input: polar gird G
Output: obstacles containing cells O

```
O ← []
for cell c in G do
  if c.getNumPoints() ≤ min_number_points then
    continue
  end if
  candidates ← 0
  candidate ← G.getLeftNeighbor(c)
  if candidate ≠ empty then
    candidates++
  end if
  candidate ← G.getUpNeighbor(c)
  if candidate ≠ empty then
    candidates++
  end if
  candidate ← G.getUpLeftNeighbor(c)
  if candidate ≠ empty then
    candidates++
  end if
  if candidates != 0 then
    addCellToObstacle(c, O)
  else
    addNewObstacle(c, O)
  end if
end for
```

As described in Algorithm 3 and graphically illustrated in Figure 4.3a, all cells of the grid are traversed line by line from the top-left corner to the bottom-

right corner of the grid. For each cell of the polar grid it is checked whether there is a left, up or up-left cell neighbor containing points. If there is a valid neighbor, the cell is added to the obstacle it belongs to, otherwise a new obstacle is created and added to the list. The result of the merging algorithm can be seen at figure 4.3b, where finally two obstacles were separated (green and blue).



(a) Traversing the grid and check for valid neighbors.

(b) Final result of merging algorithm.

Figure 4.3: Cell merging algorithm.

After all obstacles are successfully extracted, they have to be filtered. Therefore, if the number of cells of an obstacle is not within a certain range, the obstacle is removed from the list, since we want to get rid of very small obstacles, but also of big obstacles as for example walls to ease the future tracking process. Furthermore, if the obstacle height is lower than a certain value or the distance between the ground and the lowest point of the obstacle is higher than a certain value, they are also removed and not considered in the future process. For this thesis it is assumed that dynamic obstacles are always moving on the ground e.g. people or forklifts.

The overall result of the detection process are obstacles. Each individual obstacle contains a list of all cells which are belonging to the obstacle, where each cell consists of points, associated by the merging Algorithm 2.

The final obstacles detection pipeline is shown in Figure 4.4.

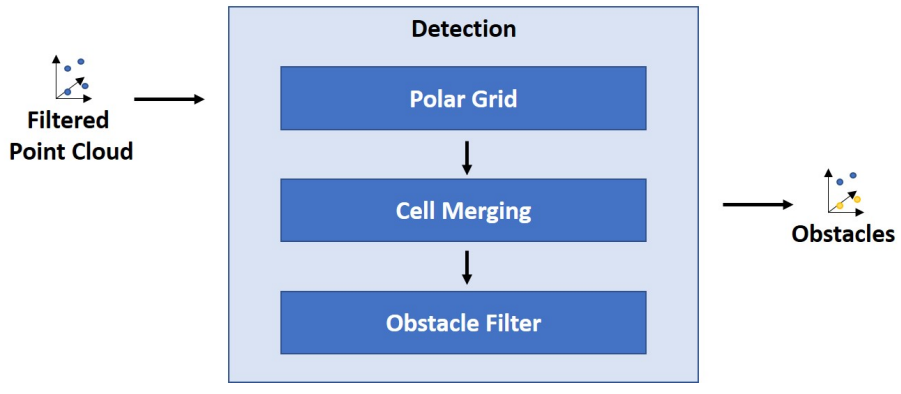


Figure 4.4: Detection pipeline.

4.2.3 Tracking

Directly after the obstacle detection, the obstacle tracking is done, to estimate information about position, speed and direction of the individual obstacles.

The obstacle tracking procedure described in the following can be rawly separated into the parts Hungarian algorithm (data association), Kalman Filter, track management, and dynamic obstacle filter.

The interconnection and the interaction of the above mentioned components is illustrated in Figure 4.5.

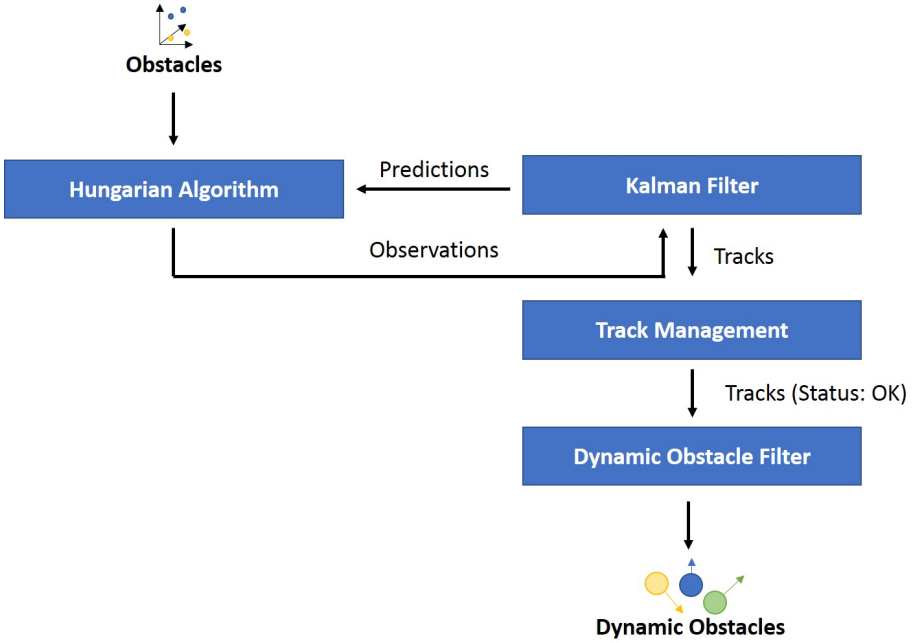


Figure 4.5: Overall tracking process.

In the following descriptions, a track is a container including all the data belonging to the Kalman Filter and further also some additional information as for example the track status.

Kalman Filter

The Kalman Filter is used exactly the way as it is described in 2.4 and all predictions are based on a constant velocity model presented at Equation 2.17.

The state vector of an observation Z is defined as:

$$Z = \begin{pmatrix} x \\ y \end{pmatrix} \tag{4.2}$$

, where x and y are the coordinates of the center of gravity calculated from all points of an obstacle. The z coordinate of the points is ignored for the tracking, since there is the assumption, that all dynamic obstacles are moving on the

ground.

At this point there has to be mentioned that the tracking is done in the world frame to deal with the ego motion of the robot. Thus, after the center of gravity is calculated, the 2D point is transformed from the sensor frame to the world frame, based on the descriptions given in Section 2.1.5.

The estimation state X and therefore the state of interest which is retrieved from the Kalman Filter can be written as following:

$$X = \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} \tag{4.3}$$

,where x and y are the estimated position coordinates and v_x and v_y are the associated velocities, based on the constant velocity model presented at Equation 2.17.

Hungarian Algorithm

Directly after the prediction step and before the update/correction step of the Kalman Filter, data association has to be done. Additionally to the predictions of the Kalman Filter, also the before calculated center of gravities of all obstacles (observations) are used as input for the association process. For this thesis a Global Nearest Neighbor approach in combination with the Hungarian algorithm, as already described in Section 2.5.2, is used to solve the association problem.

After the right associations have been found, the update step of the Kalman Filter is triggered for all tracks.

Track Management

Track management is beside the actual tracking and data association algorithm directly influencing the output of the overall tracking system. Track management includes the topics creation of new tracks, track maintenance, and track deletion.

In our system, a track can have one of the following three states:

- UNDER_INVESTIGATION
- OK
- OCCLUDED

and is updated and handled by a state machine.

At each iteration, when an observation cannot be associated to a track, if there is a new untracked obstacle in the environment, a new track is created. A new track is always initialized with the status UNDER_INVESTIGATION. Only if the track is predicted and updated over a certain period of time, the status will be switched to OK. Otherwise if the track status is UNDER_INVESTIGATION

and no observation can be associated to the prediction, the track will be deleted immediately.

This technique is used to overcome wrong detections of the obstacle detection module and moreover to make the tracking more robust and stable. The data about position, velocity and direction are not that accurate at the first few iterations, since the Kalman Filter needs a few iterations to converge.

The investigation time has a direct influence on the overall result of the tracking system. If the time is set to a high value, the system is not very reactive to changes in the environment, since it takes a long time till the state of the track is changing to OK. On the other hand, if this time is too low, wrong detections might be tracked and handled as valid track, which can cause problems at the navigation module.

When a track is already in the state OK and no observation can be associated to update/correct the prediction, the status of this track will be switched to OCCLUDED. If the status of the track is OCCLUDED for a longer period of time, the track will be deleted. Occlusions might be the reason why an obstacle is moving out of the sensor range, or it is occluded by another obstacle. Otherwise, if an observation can be associated to a track with status OCCLUDED, it is switched back to OK.

For better understanding this procedure is illustrated as a decision tree at Figure 4.6.

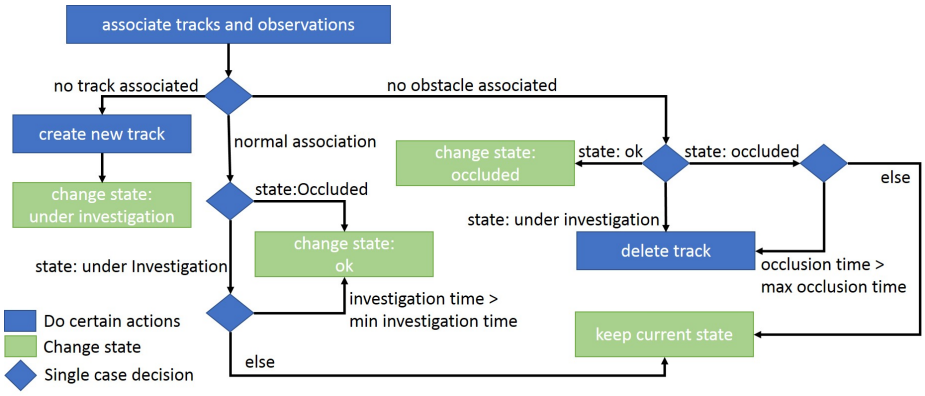


Figure 4.6: Visualization of the track management.

Dynamic Obstacle Filter

In the last step a dynamic obstacle filtering has to be done to decide whether an obstacle is dynamic or static. This step is actually not directly related to the tracking procedure, but will anyway explained here, since we are only interested in dynamic obstacles and not in static ones.

Thus, after the tracking the decision has to be made if an obstacle is dynamic and should be passed on to the navigation stack or if the obstacle is static and should be ignored for further computations. This is done using the method presented in Algorithm 4.

Algorithm 4 Dynamic Obstacle Filter

Input: tracks of the current iteration, dynamic obstacles of the previous iteration, time difference between the iterations

Output: updated dynamic obstacles

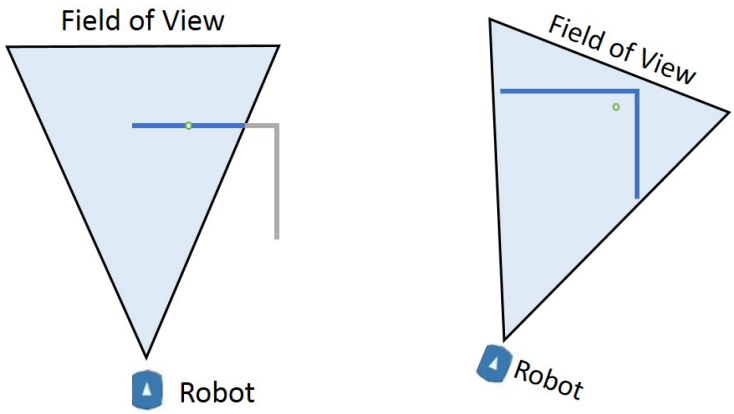
```
for track t in tracks do
  dynamic_obstacle←findObstacleAccordingToTrack(prev_dynamic_obstacles)

  if dynamic_obstacle == NULL then
    dynamic_obstacle ← createNewDynamicObstacle(t)
  end if
  if t.velocity ≥ min_dynamic_velcoity then
    if dynamic_obstacle.status == STATIC then
      dynamic_obstacle.decay -= time_difference
    end if
    if dynamic_obstacle.decay ≤ 0 then
      dynamic_obstacle.status ← DYNAMIC
    end if
  else if dynamic_obstacle.status == DYNAMIC then
    dynamic_obstacle.status ← STATIC
    dynamic_obstacle.resetDecay()
  end if
end for
```

As input to Algorithm 4, all tracks of the current iteration, dynamic obstacles of the last iteration and the time difference between the current and the previous iteration are given. The output is about a list containing all dynamic obstacles of the current iteration.

Since the tracking approach is based on the center of gravity, shape changes of obstacles will also change the position of the center of gravity. Especially for rotation this can cause short movements or leaps of the center of gravity. Thus, to the tracker it looks like the obstacle is moving and the calculated velocity is not zero. This behaviour is shown at Figure 4.7a and 4.7b, where the cone is representing the robots field of view, the green dot is representing the center of gravity, the blue line is the part of the obstacle which is inside the field of view and the gray line is the part of the obstacle outside the field of view. In Figure 4.7a the obstacle is only partly in the field of view of the robot, but as illustrated in Figure 4.7b, if the robot is turning, also the center of gravity is moving.

To reduce false positive labelling of dynamic obstacles because of short fake movements, a decay is introduced. This means, the obstacle has to move for a certain period of time with a velocity bigger than a predefined threshold until it is labelled as dynamic. The function $findObstacleAccordingToTrack()$ can be resolved, by using the concept of unique id's for both tracks and obstacles.



(a) The obstacle is only partly in the field of view of the robot.

(b) The entire obstacle is inside the field of view of the robot.

Figure 4.7: Center of gravity movement of an obstacle due to robot turning.

4.3 Dynamic Navigation

This section of the chapter introduces the already existing navigation stack and the concept which is used to include dynamic obstacles into this stack. Since the above mentioned navigation stack is actually provided by a company, it is only briefly described for reasons of protection of intellectual property.

The current navigation is based on a local planner combined with a global planner. The global planner is used to calculate a global path from the current position of the robot to a certain goal, based on previously recorded maps and a deterministic search approach. This global path is the foundation of the local path. Additionally to the global path, also other sources, as for example sensor readings are influencing the final local path.

Basically, possible trajectories, the robot is able to drive within a given time horizon, are investigated at each iteration. These trajectories are ranked based on costs, which are the result of a modular cost system. Modular means, that different cost sources are influencing the overall costs and trajectory selection. Such a cost source can be for example the global path. If it is preferred to drive as near as possible to the global path, based on a cost function trajectories near the global path are associated with low costs and otherwise with high costs. On the other hand, costs can be also retrieved from other sources, as for example a cost map containing sensor readings. Finally, these costs are added up for each trajectory.

As a practical example, trajectories near the global path do have low costs and trajectories which are overlapping with sensor readings (any kind of obstacle) do have high costs. Further, every single module (global path, sensor cost map, etc.) can be weighted and is influencing the trajectories in a different way. Finally, the best ranked trajectory, based on the combination of the costs of all cost modules is giving the local path.

Thus, the dynamic obstacle avoidance can be added to the already existing navigation stack, by adding another cost source to all trajectories.

4.3.1 Velocity Obstacles

To avoid dynamic obstacles, the velocity obstacles approach already described in Section 3.3.1 is used. The input to this algorithm is the set of all dynamic obstacles, given by the dynamic obstacle filter, where the position, the current velocity, and the direction are directly used to create or maintain new velocity obstacles.

Thus, at every iteration, already created velocity obstacles are going to be update according to the newly delivered data. The association between already created velocity obstacles and dynamic obstacles is done with the help of unique id's. If there is a new dynamic obstacle which cannot be associated with a velocity obstacle, simply a new velocity obstacle with a new unique id is create.

Due to the noise in the data of the stereo vision system, the size of an obstacle cannot be determined very accurately. Since the size of the dynamic obstacle represents the diameter of the velocity obstacle and is therefore needed for the calculations, the size of all velocity obstacles is set to a reasonable general static value.

4.3.2 Cost Calculation

As mentioned above, the cost calculation for each individual trajectory is a combination of all costs given by the different cost modules.

The costs for the global path as well as for different costmaps are already calculated by the provided navigation stack. The velocity obstacle approach is not generating costs directly as a result. Thus, a new approach to calculate costs for trajectories in relation to velocity obstacles is described below.

Since a trajectory is the path of a moving object through space as a function of time, in theory a trajectory is continuous. In the provided navigation stack on the other hand, a trajectory is described as a set of discrete points (positions) and velocities for different time stamps within a defined planning horizon. The entire calculation of kinematically possible trajectories is already done by the provided system and trajectories are just a further input to the cost calculation module.

Based on the descriptions in Section 3.3.1, collisions can be determined, by checking whether the robot velocity is inside the collision cone or not, a very basic and hard cost calculation can be retrieved by the following equation:

$$costs = \begin{cases} max\_costs \textbf{ if } \text{in collision cone} \\ 0 \textbf{ otherwise} \end{cases} \tag{4.4}$$

, where maximal costs are often called also lethal costs and trajectories with lethal costs are in general not considered in the later on trajectory selection step. Note, that for simplicity reasons velocities are often represented by their end points of the velocity vectors.

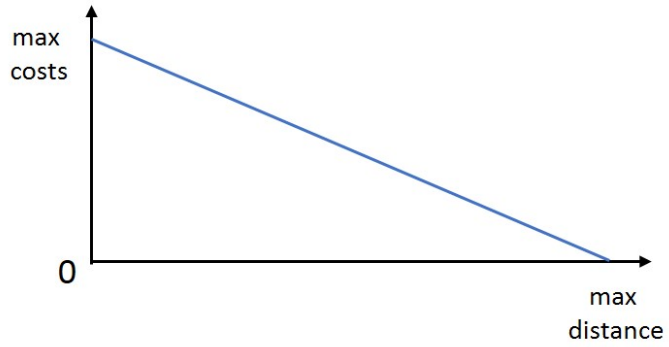


Figure 4.8: Linear cost function.

As illustrated in Figure 4.8, to smooth the driving behavior and to rate the different velocity vector end points outside the collision cone, a linear cost function based on the distance between the end point and the collision cone is defined. The costs are based on a simple linear interpolation, where the maximum distance between a velocity end point and the collision cone is equate with minimal costs and the minimum distance is equate with maximum costs. The cost calculation is illustrated in Figure 4.9 , where the blue curves are representing the available trajectories, v is the velocity, p_v is the velocity vector end point, $v_{obstacle}$ is the velocity of the obstacle and therefore the cone displacement as described in Section 3.3.1. The sounding line distance d is the distance between the collision cone and the velocity vector end point p_v. To further restrict the costs for velocity obstacles, a maximum allowed distance between the collision cone and the velocity vector end points can be defined. If the distance between the cone and the end point exceeds this defined maximum distance, the trajectory is associated with zero costs. Due to this restriction, more ideal avoidance maneuver can be achieved. Thus, with respect to Figure 4.9, the maximum distance is defined manually.

A special case occurs, if velocity vector end points are before the cone. The costs are than calculated with the help of the Euclidean distance between the point and the origin of the collision cone as presented in figure 4.10.

Finally, for safety reasons, the size of the dynamic velocity obstacle can be increased by a certain safety distance, which leads to a bigger collision cone and therefore to a further trajectory exclusion.

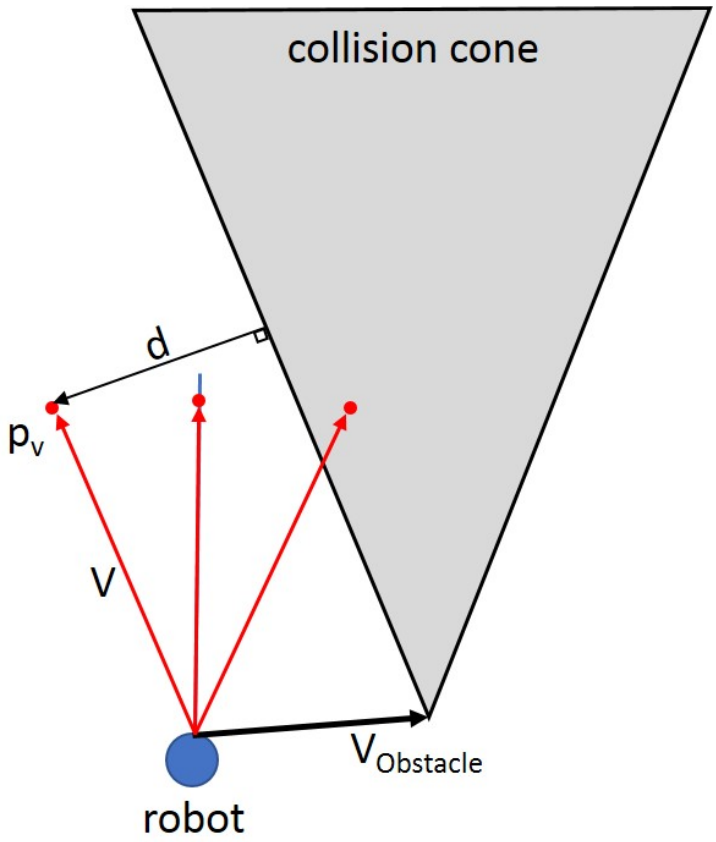


Figure 4.9: Velocity Obstacle cost calculation.

4.3.3 Trajectory Selection

As mentioned above, the costs of the different cost modules are weighted and summed up for each trajectory. Finally, the trajectory with the lowest costs is selected and executed. Note that all trajectories with lethal costs will lead to a collision anyway and are therefore not considered in the selection process.

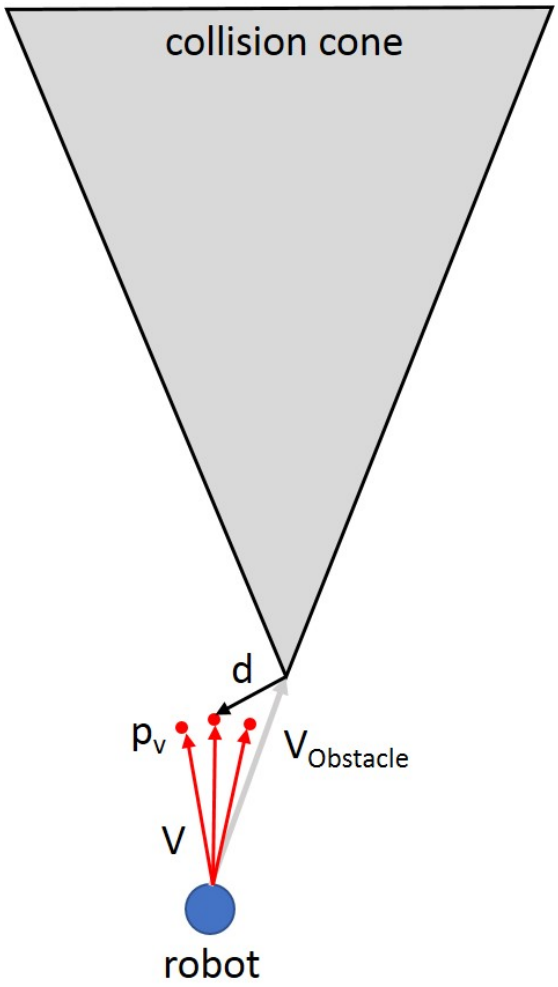


Figure 4.10: Collision cone before robot.

Chapter 5

Implementation

This chapter is used to present all hardware components which where used for the practical realization of the thesis. Such hardware components are for example computational units, sensors, etc. Further, all software packages and libraries which are needed to implement the concept described in the chapter 4 before and to do the alter on presented evaluation are stated.

5.1 Hardware

In the following sections, hardware components used for the practical realization of the concept such as the robot platform, the stereo vision system and a 3D range sensor are going to be described.

5.1.1 Robot Platform

The fundamental navigation system and the mobile robot platform is provided by the company incubedIT [1]. incubedIT is a company located in Hart bei Graz and known for developing software for autonomous navigation of mobile robots. This software is used for different platforms, manufactured by different companies.

The platform used for this thesis is called Agumos Q40 and is produced by the company Melkus. [2]. An image of the work underlying vehicle can be seen in Figure 5.1. Such vehicles are mainly used in industry within warehouses and shop floors.

The above mentioned platform provides all the functions which are needed for autonomous navigation. Thus, sensors, wheels, motors, as well as a computational unit and many more components are mounted on the platform. The computational unit, the heart of the system, is a simple industrial computer of the type Tank-700, on which are running the Robot Operating System and all nodes which are needed for the calculations.

[1] http://www.incubedit.com , 02.10.2018
[2] https://melkus-mechatronic.stadtausstellung.at/, 02.10.2018

Figure 5.1: Melkus Agumos Q40

Hardware characteristics of the Tank-700 are listed in Table5.1.

CPU	Intel Core i5 2,5 GHz
Cores	4
RAM	4 GB DDR3
Storage	SSD 80 GB
Graphics	None

Table 5.1: Characteristics of the computational unit.

Further the libraries mentioned in Table 5.2 were installed and used on the industrial computer.

Operating system	Ubuntu 14.04.5 LTS
Robot Operating System	indigo
Point Cloud Library	1.7.1
Eigen	3.2.0-8
OpenCV	3.1.0 (ros-indigo-opencv3)

Table 5.2: Libraries and versions.

5.1.2 Sensors

In this section, sensors which are explicitly used for the realization of this thesis are presented and described. Beside the stereo vision system also a 3D range sensor is described, since it is used for additional evaluations. All sensors which are by default installed on the platform are not explicitly mentioned here, since they are not directly used by the presented concept. These sensor, as for example a 2d range scanner, are used by default for the main navigation, but not for detecting dynamic obstacles.

Stereo Vision System

The stereo vision system is a combination of the Nvidia Jetson TX2[3] and the stereo vision camera ZED produced by the company StereoLabs[4].

An image of the camera can be found in Figure 5.2 and the characteristics of the camera are presented in the Table 5.3:

Video modes	2.2K, 1080p, 720p, WVGA
FPS	15 (2.2K), 100 (WVGA)
Output resolution	4416x1242 (2.2K), 3840x1080 (1080p), 2560x720 (720p), 1344x376 (WVGA)
Depth resolution	same as output resolution
Depth range	0.5 - 20 m (based on resolution)
FOV	90°(H) x 60° (V)
Size	175 x 30 x 33 mm

Table 5.3: StereoLabs ZED characteristics[4].

Figure 5.2: StereoLabs ZED stereo vision camera[4].

The Nvidia Jetson TX2 is a artificial intelligence supercomputer computer which is especially designed for computer vision and machine learning related tasks. The characteristics of this device are listed in Table 5.4.

For this thesis, the Nvidia Jetson TX2 is needed to run the ZED camera, because the indispensable SDK provided by StereoLabs is based on CUDA and therefore a GPU is needed which is not installed in the transport robot. To retrieve the camera data in form of a message over a ROS topic within a node, the zed_ros_wrapper package is used [5]. Following versions of the above mentioned libraries and packages where installed on the Nvidia Jetson TX2:

[3]https://www.nvidia.com/de-de/autonomous-machines/embedded-systems-dev-kits-modules/, 02.10.2018

[4]https://www.stereolabs.com/zed/, 02.10.2018

[5]http://wiki.ros.org/zed-ros-wrapper, 02.10.2018

GPU	NVIDIA Pascal, 256 CUDA-Cores
CPU	HMP Dual Denver + Quad ARM A57
RAM	8 GB, 128 Bit-LPDDR4
Storage	32 GB eMMC
Used connections	USB 3.0, Gigabit Ethernet

Table 5.4: Characteristics of the Nvidia Jetson TX2[3].

Operating System	ubuntu 16.04.4 LTS
JetPack	3.2
Robot Operating System	kinetic
ZED SDK	2.3.3
CUDA	9.0
zed-ros-wrapper[5]	2.3

Table 5.5: Libraries needed to run the ZED stereo vision camera.

Finally the Nvidia Jetson TX2 was mounted inside the robot corpus and the mounting of the StereoLabs ZED camera can be seen at Figure 5.1.

Velodyne VLP-16 Puck

In this section the result of this work is a little bit anticipated. Based on the low range and low rate data is delivered by the stereo vision system in combination with ROS, additionally a 3D range sensor is used to evaluate the work done for this thesis. As described in detail at Section 6.2.1, the low data rate is due to the high resolution which is needed to increase the depth range, where the camera depth range is directly related to the camera resolution. The higher the resolution the higher the depth range. The mentioned sensor is the well known Velodyne VLP-16 Puck presented in Figure 5.3 with following characteristics: [6]

Horizontal FOV	360°
Vertical FOV	±15°
Range	100m
Channels	16
Accuracy	± 3 cm
Rotation rate	5 - 20 Hz
Angular resolution	vertical: 360° horizontal 0.1° - 0.4°

Table 5.6: Characteristics of the Velodyne VLP-16 Puck[6].

To run this sensor, the driver package presented at https://github.com/ros-drivers/velodyne is used.

[6]https://velodynelidar.com/vlp-16.html, 02.10.2018

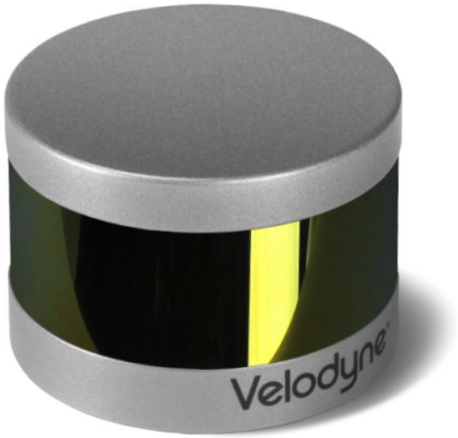


Figure 5.3: Velodyne VLP-16 Puck 3d laser range finder[6].

5.2 Infrastructure

This section is used to illustrate the connections and the collaboration between the different hardware devices as well as the communication between the individual nodes.

5.2.1 Hardware Infrastructure

In Figure 5.4 the connections of the hardware devices mentioned above are illustrated.

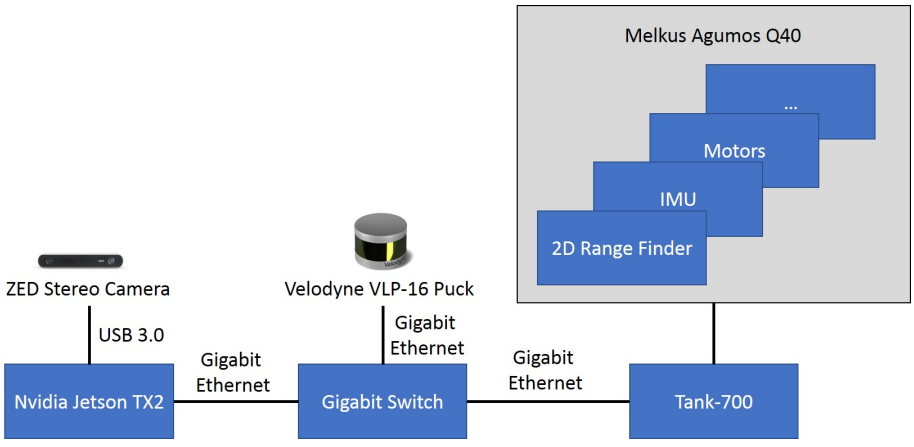


Figure 5.4: Hardware structure.

In Figure 5.4 it can be easily seen, that the ZED stereo camera is directly connected to the Nvidia Jetson TX2 over an USB 3.0 interface. This device is further connected to a gigabit ethernet switch. Further, also the Velodyne VLP-

16 Puck and the Tank-700 are connected to this switch. The default sensors as for example a 2D range scanner, an IMU and the motors of the platform are also connected over different interfaces to the Tank-700.

5.2.2 Node Infrastructure

To implement the concept described in Section 4 several nodes are needed. These nodes and the message types of their communication channels (topics) are illustrated at figure 5.5.

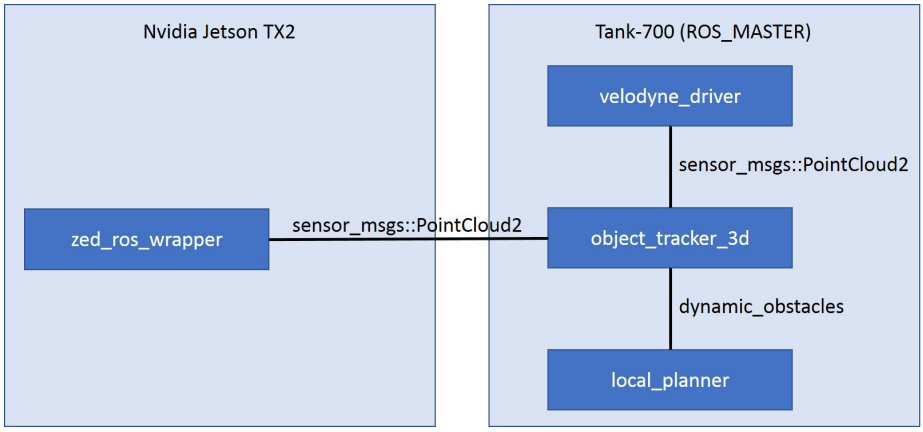


Figure 5.5: Node structure.

The Nvidia Jetson TX2 is only used to run the node zed_ros_wrapper[5]. This node is nothing else then the zed camera driver, which is providing the 3D point cloud in form of the standard ROS message *sensor_msgs::PointCloud2* to the rest of the system.

The ROS_MASTER and all other nodes needed to run the navigation stack and the detection system are executed on the Tank-700. Thus the Tank-700 is the main computational unit of the system and the Nvidia Jetson TX2 can be seen as a support unit.

The node obstacle_tracker_3d is responsible for all the calculation belonging to dynamic obstacles, as described in Section 4.2. The node local_planner is responsible for all the navigation calculations, described in Section 4.3.

To enable the communication between the obstacle_tracker_3d node and the local_planner node, a specific message type called *dynamic_obstacles* is introduced. This message contains a list of all dynamic obstacles and is defined in Listing 5.1, where *dynamic_obstacle* is again a specific message type defined in Listing 5.2.

```
1 // dynamic_obstacles.msg
2
3 Header header                  // standard ros header
4 dynamic_obstacle[] obstacles   // list of all dynamic obstacles
```

Lst. 5.1: Specific dynamic obstacles message.

```
// dynamic_obstacle.msg

float x       // x position in global frame
float y       // y position in global frame
float theta   // orientation in global frame
float vx      // velocity in x direction in global frame
float vy      // velocity in y direction in global frame
```

Lst. 5.2: Specific dynamic obstacle message.

The position (x,y) as well as the orientation and the velocity components are given in the global frame, since the tracking is done in the global frame and the data is directly coming from the tracking system.

In short, one can say that each *sensor_msgs::PointCloud2* message delivered by the StereoLabs ZED camera or by the Velodyne VLP-16 Puck laser scanner is processed in the node obstacle_tracker_3d and dynamic obstacles are published if form of *dynamic_obstacles* message.

Chapter 6

Evaluation

In this section we present the results of an experimental evaluation of the system developed.

First, the dynamic obstacle detection, especially the tracker is evaluated. Therefore, quantitative results like accuracy and also qualitative results (discussion of special cases) are presented. After the evaluation of the tracker, an evaluation of the influence of using dynamic obstacles on the performance of the navigation system based on simulation data is presented. At the end an evaluation considering human factors can be found. For this evaluation human factors are about feelings and subjective opinions of probands according to the navigation systems. Therefore, the original navigation system and the new navigation system are operated in a real world scenario and the opinion and feelings of a group of people are collected to rate the different approaches. For simplicity, in the sections below the navigation system provided by the company incubedIT is always mentioned as reference navigation system.

6.1 Environment

This section describes the physical environment, which is an indoor robot test area, and the tools, which were used to do the evaluation. As described above, a simulation as well as real world scenarios are used to perform a quantitative and a qualitative evaluation of the work done for this thesis.

6.1.1 Simulation

The simulation used for the evaluation is a further resource provided by the company incubedIT. Since the reference navigation system is only based on a simple 2D range scanner, also the simulation is only providing 2D data and not 3D data. But anyways this 2D data is enough to evaluate the new concept of dynamic navigation presented in Section 4.3, since this data has ground truth. To enable dynamic obstacles in the simulation, two simulated robots are used, were one robot is running the navigation algorithms and the other one is simulating the dynamic obstacle. The dynamic obstacle is simple following a strict path between two predefined goals and is not avoiding obstacles in any sense. A sample setup can be seen at Figure 6.1 , where the blue polygon with the label

sim126 is representing the robot and the polygon with the label sim128 is representing the obstacle. The white triangle within the blue polygons is showing the orientation of the robot, the purple line represents the path given by the global planner and the orange boxes represents goals, the robot can drive to.

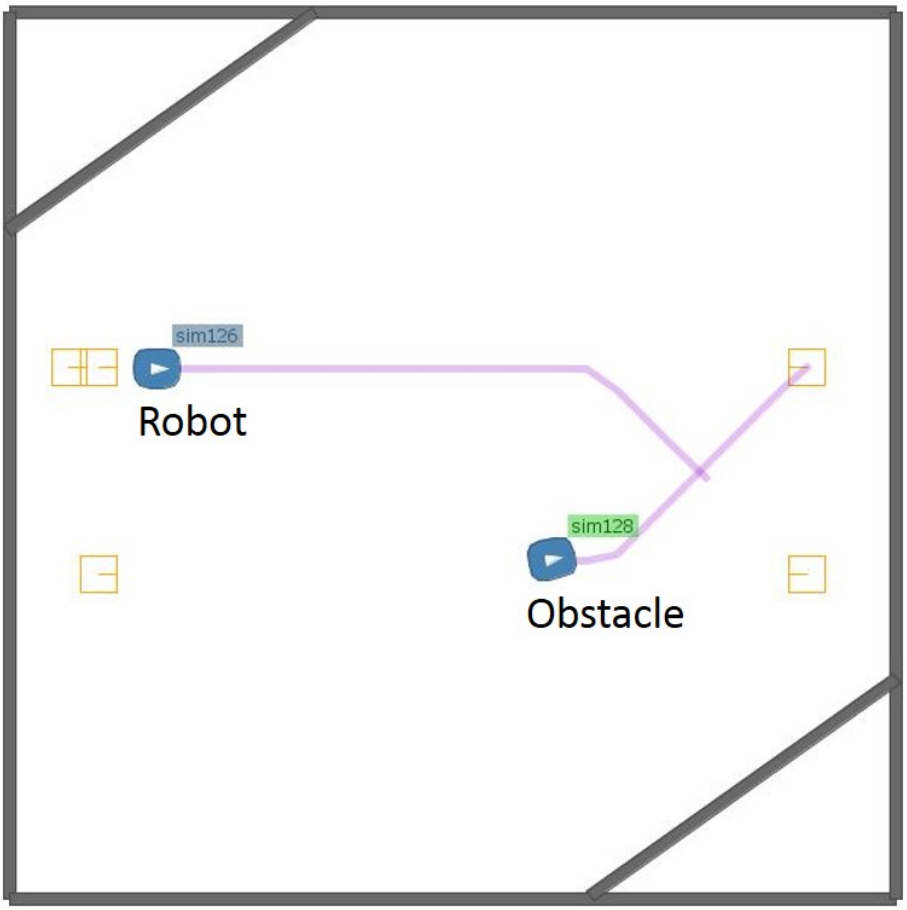


Figure 6.1: incubedIT simulation.

6.1.2 Motion Capture - Optitrack

To evaluate the tracking system and therefore the dynamic object detection procedure, a ground truth is needed. Thus, for this evaluation the data given by the tracking system is compared to the data given by a motion capture system called OptiTrack [1].

A big benefit of OptiTrack lies within the fact that it is ROS compatible and the data can be simply used within a ROS system using the package mocap_optitrack [2].

[1] https://optitrack.com/, 02.10.2018
[2] http://wiki.ros.org/mocap_optitrack , 02.10.2018

To track a specific object with the Optitrack system, at least three so called tracking markers has to be place on the object. This tracking markers are tracked by cameras mounted at the ceiling and finally the Optitrack node is publishing the 2d pose and the velocity of the tracked object.

6.2 Tracking Evaluation

This section of the chapter presents the evaluation results of the tracking system which is in detail described in Section 4.2.3.

The evaluation of this module is split up into the two sections quantitative evaluation and qualitative evaluation. Both are described in detail below.

6.2.1 Quantitative Evaluation

To evaluate the accuracy of the tracker, the position as well as the velocity estimated by the tracking system are compared to the ground truth data given by the OptiTrack system.

As a recap, the position of a dynamic obstacle is represented by the x and y coordinates of the center of gravity in the global frame. Thus position p and velocity v can be represented by $p = (x, y)$ respectively $v = (v_x, v_y)$.

As metric for the comparison itself, the Gaussian Normal Distribution of the Euclidean distance between the position p, the components x and y, as well as v, v_x and v_y of the tracker and the OptiTrack system is chosen. In the following sections the Gaussian Normal Distribution for all individual components is investigated and illustrated.

At this point of the evaluation there has to be mentioned, that the entire system is not only evaluate on 3D data delivered by the stereo vision system, but also on data delivered by a 3D laser range scanner. The additional sensor is needed, since the stereo vision system can be only used in the video mode WVGA to ensure real time computations. Running the camera on such a low resolution mode causes a drastic reduction of the detection range. Based on the methods used within the described concept and running the camera in WVGA mode, the overall system is able to detect obstacles which are up to 4 meters away from the sensor. This range is too short to do robust dynamic obstacle avoidance, since the data contains always noise and the methods used within this concept do not guarantee to detect dynamic obstacles at the first instance. If the dynamic obstacle is detected too late, the robot is not able to execute a proper avoidance maneuver.

Higher resolution of the camera on the other hand means higher range, but also longer duration for the computation. Therefore, real time usage is not guaranteed anymore. To run the reference navigation system, data which is necessary for the navigation stack must be provided with at least 8 Hz. Since our proposed navigation system is an extension of the reference navigation system, also the new used sensors and especially the whole detection process should provide data with a rate of at least 8 Hz to the navigation stack.

As it is presented in the following sections, the results based on the ZED camera in mode WVGA are not satisfying. Therefore, we searched for another

sensor which meets our requirements and came up with the Velodyne VLP-16 Puck.

In the following Table 6.1 the average frequencies for the two different camera resolutions WVGA and 720p as well as for the Velodyne laser scanner are presented.

Sensor/Resolution	**Frequency**
ZED WVGA	13.8 Hz
ZED 720p	3.9 Hz
Velodyne	9.9 Hz

Table 6.1: Data frequencies for different sensors.

To obtain the positional deviation as well as the velocity deviation between the data given by the tracking system and the OptiTrack, two specific scenarios are created.

The first scenario is about a static evaluation of the tracker. Therefore an observer is placed on a fixed position and the dynamic obstacle is driving exactly four times between two goals with a maximum speed of one meter per second. This scenario is illustrated in Figure 6.2 below, where it clear to see that the robot is not only doing linear movements but also slightly curves.

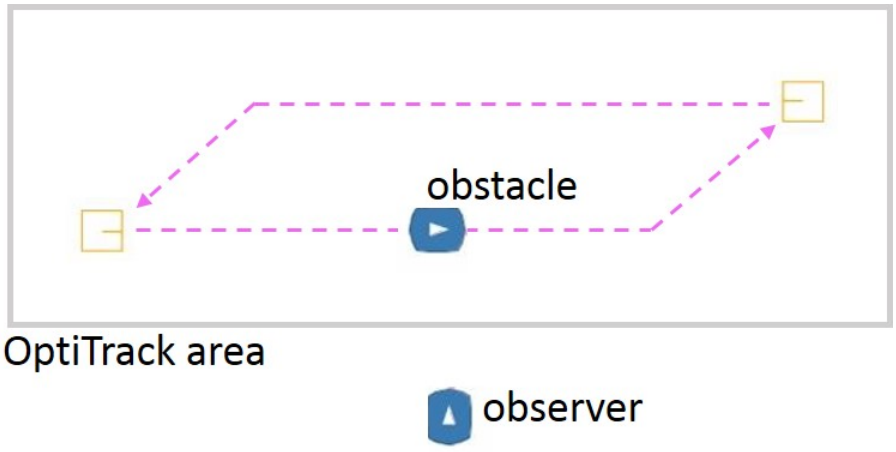


Figure 6.2: Static evaluation of the tracker.

Further, there is also a dynamic evaluation of the system performed. Dynamic means, that not only the obstacle is moving, but also the observer is moving.

Note, that the tracking results are directly depended on the localization of the robot, since the tracking itself is done on the global frame. Thus, all transformations, which has to be done to transform the points from the sensor frame to the global frame are depended on the accuracy of the localization. Since the goal is to evaluate the tracking system in combination with the already existing system, the error propagation of the localization to the tracking system is not explicitly investigated at this point. It is absolutely true, that the error

of the localization is influencing the overall error of the tracking system. The scenario used for the dynamic evaluation is illustrated in Figure 6.3.

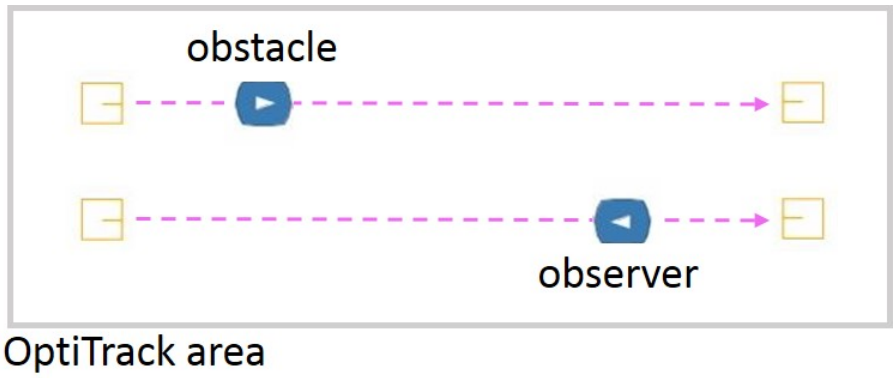


Figure 6.3: Dynamic evaluation of the tracker.

The positional standard deviation for the static scenario as well as for the dynamic scenario are listed in the Table 6.2. As it can be seen in Table 6.2 and 6.3, there is no data collected for the dynamic case based on the ZED camera. This is due to the limited detection range of the camera. Since the obstacle as well as the observer are moving towards each other, there is not enough time for the tracking system to start tracking the obstacle.

		ZED [m]		**Velodyne [m]**	
		mean	std	mean	std
Static	x	0.182	0.074	0.138	0.091
	y	0.086	0.085	0.281	0.086
	position	0.213	0.087	0.325	0.091
Dynamic	x	-	-	0.202	0.100
	y	-	-	0.165	0.142
	position	-	-	0.306	0.064

Table 6.2: Positional deviation of the tracking system compared to the motion capture system OptiTrack.

As it can be easily seen in Table 6.2, the difference of the standard deviation for both cases are in the range of centimeters.

Interesting is the difference of the position value between the static and dynamic situation according to the Velodyne sensor. In the static case, the standard deviation for the overall position is 0.091 meters, while the dynamic is even lower with 0.064 meters. On closer inspection, however, there can be seen that the deviation for the x value as well as for the y value of the dynamic case are higher than for the static once.

While the results for the static situation for both sensors are almost the same, evaluating the dynamic situation (robot and obstacle are moving) with the camera is not even possible due to the descriptions about the resolution problem mentioned above.

The velocity deviation is presented in Table 6.3, where v is the resulting combination of the velocity components v_x and v_y. Again, the standard deviations for the dynamic evaluation are a little bit higher than for the static once. This is due to the localization error occurring while the robot is driving.

		ZED [m]		**Velodyne [m]**	
		mean	std	mean	std
Static	v_x	0.842	0.173	0.808	0.203
	v_y	0.265	0.151	0.264	0.147
	v	0.098	0.109	0.073	0.069
Dynamic	v_x	-	-	0.512	0.253
	v_y	-	-	0.057	0.060
	v	-	-	0.112	0.078

Table 6.3: Deviation of the velocity of the tracking system compared to the motion capture system OptiTrack.

As it is mentioned above, it is assumed, that the error between the motion capture system and the tracking system can be modeled with a Gaussian distribution. Therefore, in Figure 6.4 the Gaussian distribution according to the position data delivered by the ZED camera is shown and it can be seen, that the assumed model fits to the data. In the case of the overall position distribution, the standard deviation from the mean (0.213 meter) is 0.087 meter. Thus, the error model can be described as a mean shifted Gaussian distribution.

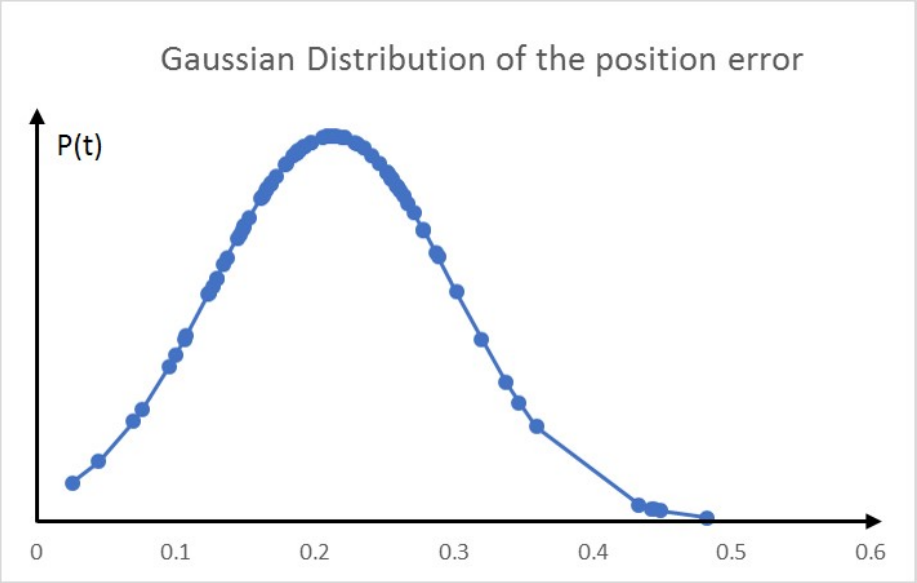


Figure 6.4: Mean shifted Gaussian distribution of the positional error.

6.2.2 Quantitative Evaluation

Beside the above described quantitative evaluation of the tracker, there was done a qualitative evaluation to get some information about the performance of the tracker according to performed special cases. Therefore, the number of new track creations and wrong data association is investigated. Such special cases are for example crossings of two tracked dynamic obstacles and occlusions of the tracked dynamic obstacles. For the following evaluation, the observer is placed at a fixed position and not moving, since this evaluation is focusing on the data association and track management while performing the special cases. For a static observer, the localization is not changing, which means that the localization error is not given.

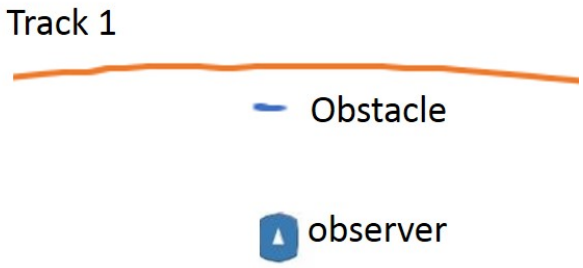
Occlusion

Occlusion in the sense of tracking simple means, that an obstacle is completely or partly occluded by any other obstacle. The difficulty lies here in the correct prediction of the occluded track. If the tracker is not able to associate an observation to the predicted track, a new track is created and the old one is deleted. Thus, tracking the occluded obstacle was not successful.

Based on the illustrations shown in Figure 6.5a and 6.5b, the presented tracking system is able to track obstacles which are occluded up to 75 percent. If obstacles are completely occluded, the tracker is failing. This test is done with a person which is 1.75 meter tall and according to the 75 percent occlusion, the occlusion height is 1.3 meter and the obstacle width is 0.9 meters.

In Figure 6.5a the orange line is representing the path of the dynamic obstacle given by the tracking system and the blue line is representing the obstacle which is partly occluding the dynamic obstacles.

In Figure 6.5b it can be seen, that two tracks are needed to track the dynamic obstacles, where the orange line and the blue line are the paths of the dynamic obstacle and the small gray line is the obstacle occluding the dynamic obstacle. Thus, the obstacle which is occluding the dynamic obstacle forces the tracking system to delete track 1 and to create a new track for the same obstacle if it visible again.



(a) The obstacle is 75 percent occluded by an object. The object has a height of 1.3 meters.

(b) The obstacle is 100 percent occluded by an object.

Figure 6.5: Track occlusion.

Crossings

To manage the tracking process while two objects are crossing is not a trivial task for a tracking system. Within such a scenario, the data association algorithm is most challenged. The situation and the performance of the tracking system is illustrated in Figure 6.6

As it can be seen at Figure 6.6, the tracker is only able to track one dynamic obstacle for the full duration of the scenario. For the other obstacle, a new track is created after the crossing. This is due to the configuration of the track manager described in 4.2.3. Since the goal for this thesis is to minimize wrong tracks and pass only data to the navigation stack which is as accurate as possible, the occlusion time of a track is set to 0.2 seconds. In other words this means, if the obstacle is occluded for more than two time intervals, the track is going to be removed. Thus, within the scenario presented in 6.6, finally 3 tracks are needed to track two dynamic obstacles.

A drawback of having multiple tracks for the same obstacle is, that the tracking ID of the obstacle is changing. For the tracking system it seems like this is a completely new obstacle. Depending on the purpose of tracking, this

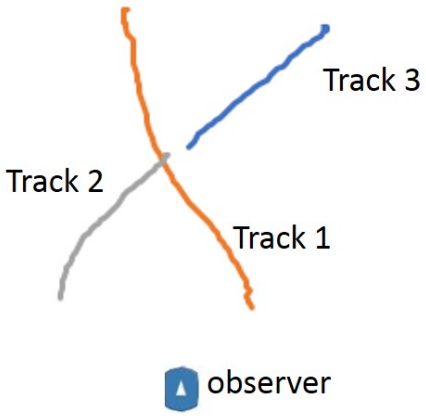


Figure 6.6: Tracking performance, while the paths of two obstacles are crossing.

might lead to errors or is just simply wrong. If the goal is to track people for surveillance, this configuration is not an option, since there should be always exactly one track and one ID per person. For this thesis the tracking is about to get velocity data and to estimate future positions to avoid dynamic obstacles. Thus, having only one track per obstacle is not a constraint. On the other side, too heavy new track creation is also bad, since the Kalman Filter needs a certain duration to converge to provide reliable data.

6.3 Navigation Evaluation

In this section the reference navigation stack is compared to the extended navigation approach described in Section 4.3.

Therefore, a quantitative evaluation based on a certain metric is presented, followed by a qualitative evaluation based on special cases.

6.3.1 Quantitative Evaluation

As described above, a quantitative evaluation is based on a certain metric. This metric is described in the following and based on the most interesting factors for the company incubedIT.

- **Collisions:**
  Sums up the number of occurred collisions during a run. The value is only incremented if the robot is driving into the obstacle. If the obstacle is driving into the still standing robot it is not counted as collision. This assumption is needed, since the simulated obstacle is treaded as a stupid agent, not avoiding any kind of collisions. Thus even if the robot stops, the obstacle will drive into it.

- **Time:**
  Time represents the duration of a test run in seconds.

- **Mileage:**
  Mileage represents the distance the robot was driving during an evaluation run in meters.

- **Goals:**
  Goals represents the number of goals, the robot was able to reach during a test run.

- **Velocity:**
  Velocity represents the avarage velociy of the robot during a test run. This value is a combination of the parameter Time and Milage, where

$$Velocity = \frac{Mileage}{Time} \tag{6.1}$$

- **Time per Goal:**
  Time per Goal is a combination of the parameter Time and Goal, where

$$Time/Goal = \frac{Time}{Goal} \tag{6.2}$$

  This value is representing the average time that is needed to reach a goal.

- **Time per Collision:**
  Time per Collision is a combination of the parameter Time and Collision, where

$$Time/Collision = \frac{Time}{Collision} \tag{6.3}$$

  This value is representing the average time between collision of the robot and the dynamic obstacle.

For the quantitative evaluation a specific scenario is used. This scenario is illustrated in Figure 6.7 and described below.

As it can be seen in Figure 6.7, the task of the robot is to drive continuously between the goals L1 and R2, while the dynamic obstacle is driving from the goal L2 to the goal R1. The distance between goal L1 and R2 as well as L2 and R1 is exactly 12.497 meters. Both, the robot as well as the dynamic obstacle are starting to drive at the same time at the goal L1 respectively L2. The robot is limited to a maximum velocity of one meter per second, whereas the dynamic obstacle is limited to a maximum velocity of 1.3 meter per second. The different velocity parameterization is done to provoke different situations. Thus, crossing situations, parallel driving situations and many others are generated and influencing the quantitative evaluation.

The final results of this evaluation are presented in Table 6.4 below.

As it can easily be seen in Table 6.4, by looking at the criterion Collisions the new proposed navigation system mainly based on velocity obstacles is avoiding dynamic obstacles in the given scenario perfectly. The robot with the reference navigation system, on the other hand, collided four times with the dynamic obstacle. Thus, approximately every 10.5 minutes, the robot is crashing into the dynamic obstacles. Based on this metric, this is the only benefit of the new proposed system in comparison with the reference system.

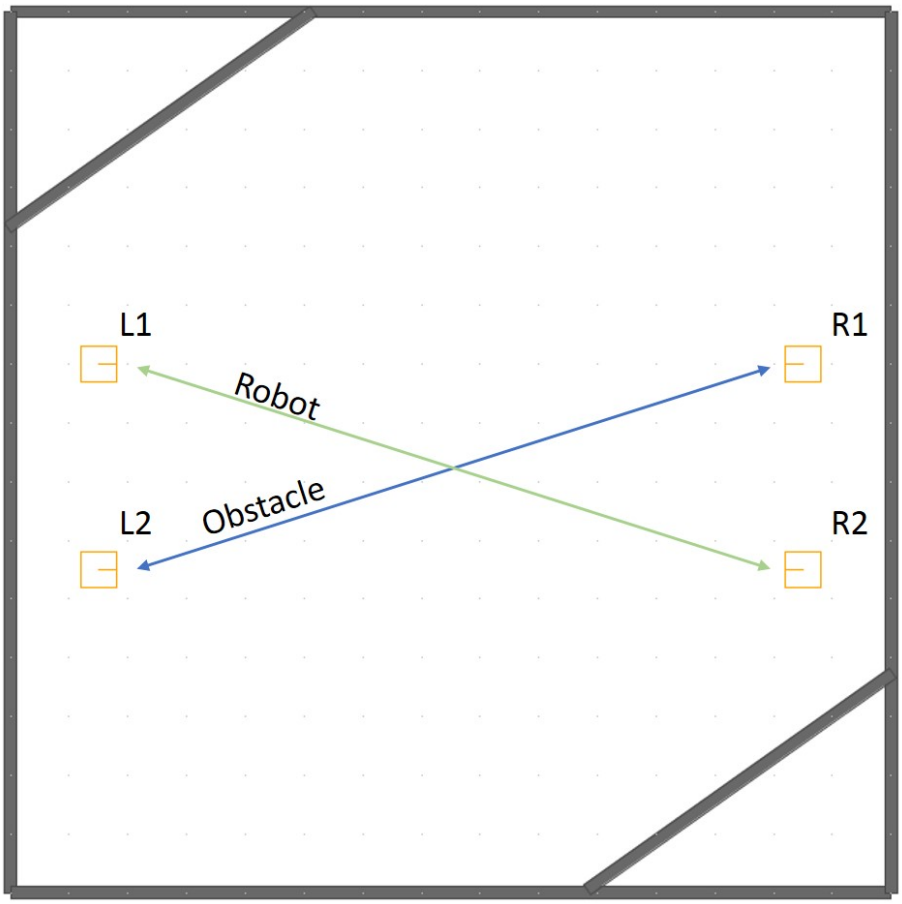


Figure 6.7: Quantitative scenario.

	reference navigation	**extended navigation**
Collision	4	0
Time	2582,57 s	3063,52 s
Mileage	1291,63 m	1373,90 m
Goals	100	100
Velocity	0,50 m/s	0,45 m/s
Time/Goal	25,8257 s	30,6352 s
Time/Collision	645,64 s	-

Table 6.4: Quantitative evaluation of the navigation systems.

The average time to reach a goal for a robot running the reference navigation system is lower than a robot running the new navigation system. This is because the distance traveled and also the time needed to reach the 100 goals is lower. With respect to shortest paths, on average the shortest path found by the local planner of the reference system is better than the shortest path found by the extended local planner. Thus, also the throughput of the reference system is

higher.

Safety and dynamic obstacle avoidance on the other hand is only enabled by the newly proposed navigation system.

6.3.2 Qualitative Evaluation

For the qualitative evaluation several special cases namely crossing, overtaking and approaching each other are investigated. For all these special cases, the evaluation is based on the comparison between the reference navigation system and the new introduced navigation system. All above mentioned scenarios are illustrated and described in the following sections below.

Crossings

Crossings in general are scenarios which happens a lot in reality and it is not that easy for the navigation system to overcome this situation with an optimal solution. Optimal in this scenario means, that the robot should drive continuously and should stop before doing an avoidance maneuver.

In this scenario, the distance between the goals N and S is exactly the same distance as for W and E, where the goals N and S are vertically aligned and the goals W and E are horizontally aligned. Thus, a collision somewhere in the center of the goals is forced. An illustration of this specific case is shown in Figure 6.8.

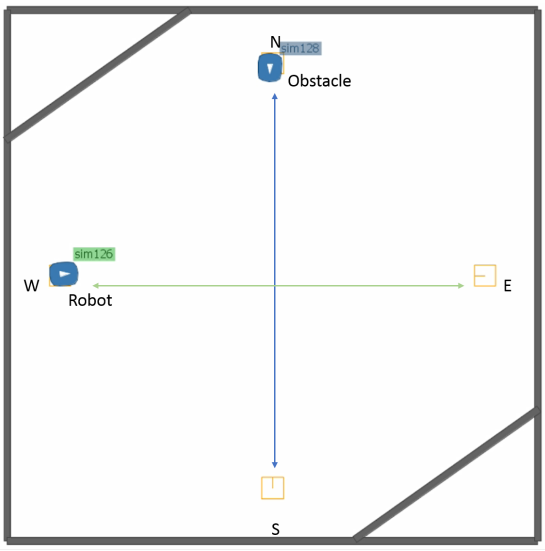


Figure 6.8: Initial situation of the crossing scenario.

As it is illustrated in Figure 6.9, the reference navigation system is not able to solve this situation in a smooth and ideal way. The robot is driving straight to the goal on the opposite and just before the robot crashes into the dynamic obstacle it stops. In order to get out of this situation the robot is turning to the right and searching for a new path. In the meanwhile, the dynamic obstacle

is already gone. The gray line in Figure 6.9 is representing the path, the robot was already driving, where the purple line is representing the path the robot has to follow to reach the goal (global plan).

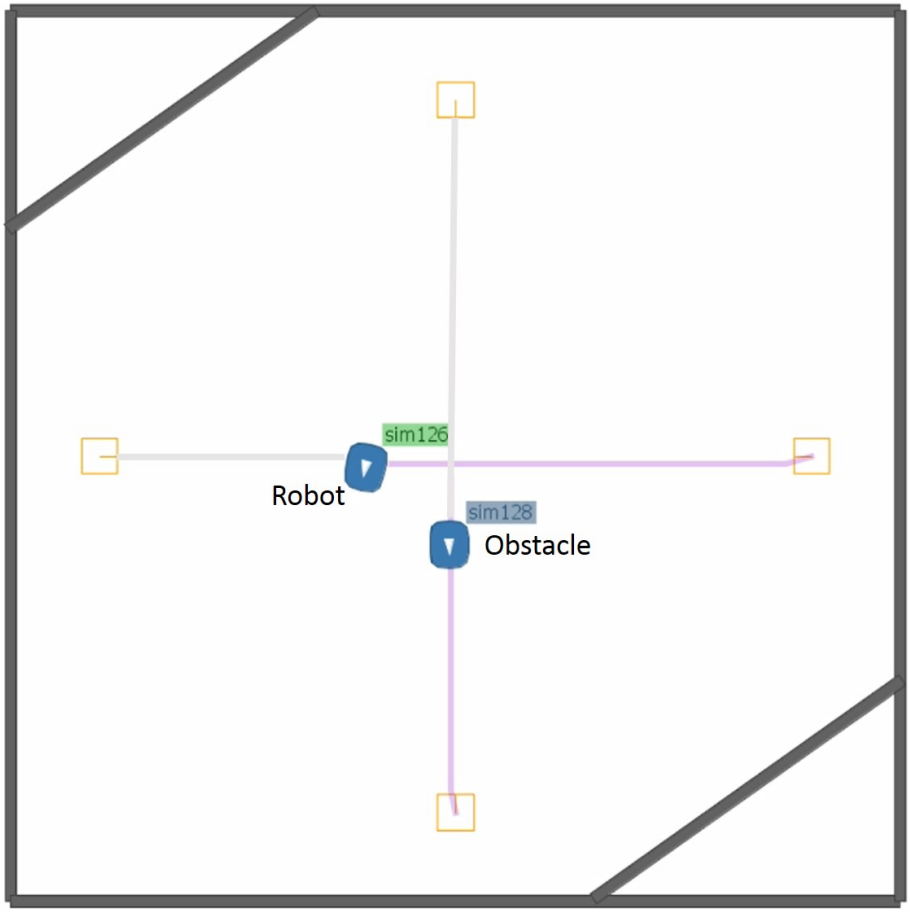


Figure 6.9: Behavior of incubedIT navigation system at crossings.

The extended navigation system on the other hand is able to overcome this situation in a smooth and elegant way, by simply reducing the speed and let the dynamic obstacle pass. After the dynamic obstacle passed, the speed is increased to the maximum again.

Overtaking

Overtaking simple means, that the robot is driving faster than the obstacle and therefore the robot has to pass the obstacle at a certain point of time. For this scenario the maximum speed of the obstacle was limited to 0.5 m/s and the maximum speed of the robot was limited to 1.0 m/s.

The reference navigation system is able to pass the obstacle, but the distance between the robot and the obstacle is about one meter as shown in Figure 6.10

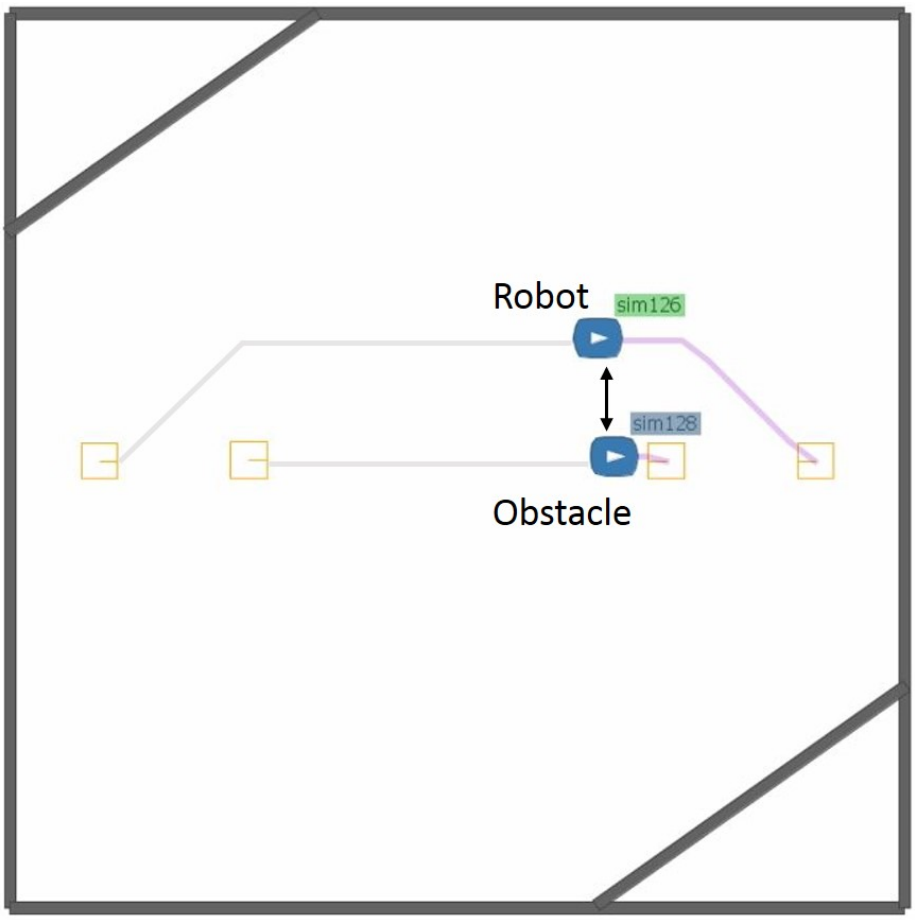


Figure 6.10: Distance between robot and obstacle during an overtaking maneuver.

The navigation approach presented in this thesis is not able to do the overtaking maneuver in a satisfying way. Due to the concept of cost calculation described in Section 4.3.2, the trajectory costs near the robot are lower than the costs near the velocity obstacle. In this scenario the costs for trajectories, which lead the robot to turn away from the dynamic obstacle are minimal cheaper. Based on the reference navigation system trajectory selection implementation the robot starts to execute the cheapest trajectories and turns away from the dynamic obstacles. Due to the underlying motion model, this turning

behavior is slow and therefore the robot needs a lot of time to finally overtake the obstacle.

Thus, the overtaking maneuver of the extended navigation system is not as smooth as the overtaking maneuver of the reference system. Further, the overtaking maneuver of the extended system needs approximately 5 seconds longer to reach the goal.

The trajectory costs for the initial situation are shown in Figure 6.11, where red colored trajectories are associated with low costs and purple colored trajectories are associated with high costs. The green triangle is representing the collision cone of the velocity obstacle and the red arrow is showing the displacement described in Section 3.3.1.

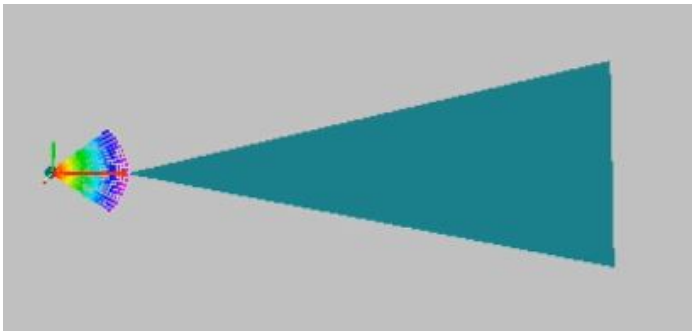


Figure 6.11: Cost representation according to the initial situation of the overtaking scenario.

At Figure 6.12, the red lines are indicating the cheap trajectories, which leads the robot to turn and to drive a little bit downwards.

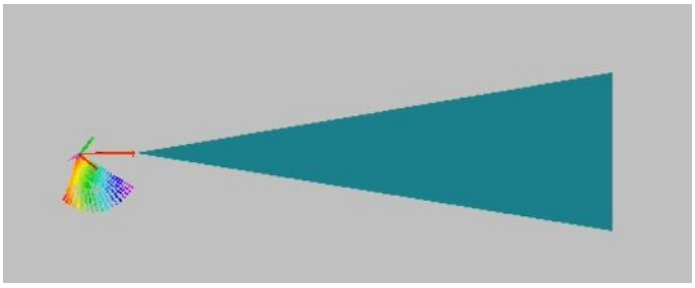


Figure 6.12: Cost representation according to the overtaking scenario.

Approach Each Other

This scenario is used to present the main difference between the dynamic obstacle avoidance maneuvers of the different navigation systems. The robot as well as the dynamic obstacle are approaching each other and the robot tries to pass the dynamic obstacle with a minimal distance and without colliding into it.

The maneuver done by the reference navigation system is pretty much the same as described in Section 6.3.2. The robot is driving directly to the goal and just before the robot crashes into the obstacle it stops. This behavior is shown in Figure 6.13, where the purple trace belongs to the robot and the gray trace belongs to the obstacle.

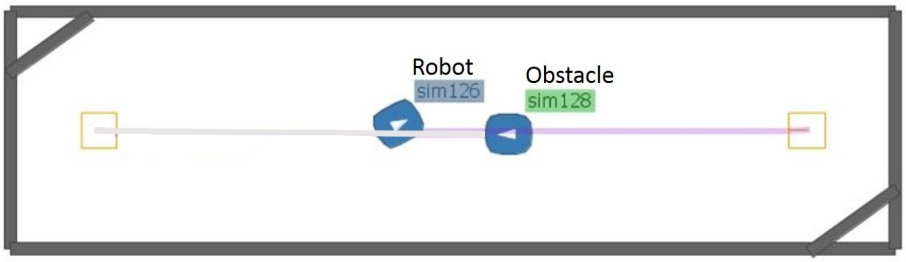


Figure 6.13: Robot is approaching the dynamic obstacle, but not able to pass.

This situation can be absolutely satisfying solved by the approach presented within this thesis. As it can be easily seen at Figure 6.14, the robot is avoiding the obstacle with a minimal distance between robot and obstacle.

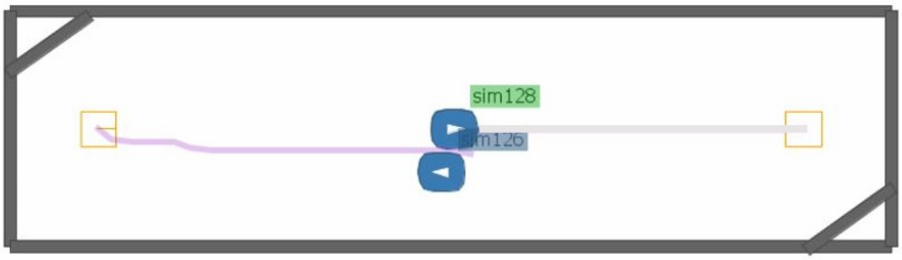


Figure 6.14: Robot avoiding the obstacle in an ideal way.

6.4 Human Factor Evaluation

As it is written in the introduction of this chapter, the overall system is additionally evaluated on human factors. The human factor evaluation is done not only to get objective information about the navigation systems, but also subjective information like feelings of the people. This kind of information is as important as objective information, since at the end of the day people have to collaborate with the robots.

6.4.1 Questionnaire

In order to be able to perform this kind of evaluation, first some key factors of interest which are related to robot navigation in general are defined. These factors are about predictability of robot movements, safety, and the general feeling of people while interacting with the robot. All the above described factors are transformed to likert-scale questions and written down to a questionnaire [2]. Further, an open ended question is added, to get even more information about the probands. This questionnaire is presented in Figure 6.15. All these questions are aimed to get information about the personals feeling and minds of the probands accordingly to the different navigation systems. Due to data protection, this questionnaire was done anonymous.

Figure 6.15: Questionnaire.

6.4.2 Setup and Data Collection

The scenario used for the evaluation is presented in Figure 6.16.

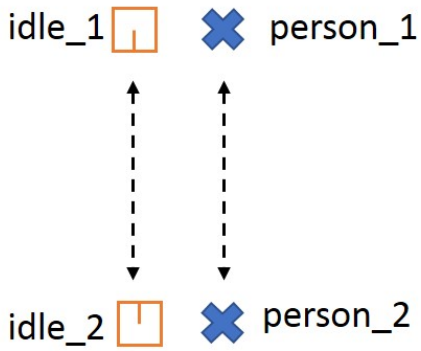


Figure 6.16: Scenario for the human factor evaluation.

The robot has to drive from goal idle_1 to goal idle_2 while a person is walking from goal person_1 to goal person_2 in a straight line. The distance between the goal of the robot and the goal of the person is only about 30 centimeters, to force the robot to react to the persons movements.

Both navigation systems were tested based on the presented questionnaire and the scenario described above. Each proband was first tested on the reference system followed by the extended navigation system, where the probands did not know, which system was currently running.

The test is based on 10 probands who have general knowledge about robotics, but are no experts in robot navigation.

6.4.3 Results

After all the data is collected, the answers to the likert-scale questions are transformed to the following ranking shown in Table 6.5:

Title	**Score**
strongly disagree	0
disagree	1
neither nor	2
agree	3
strongly agree	4

Table 6.5: Likert-scale question scoring.

To rank the two different navigation systems, the sum overall likert-questions and probands is calculated and presented in Figure 6.17. The reference system reached a score of 93 points while the score value of the new system is 150.

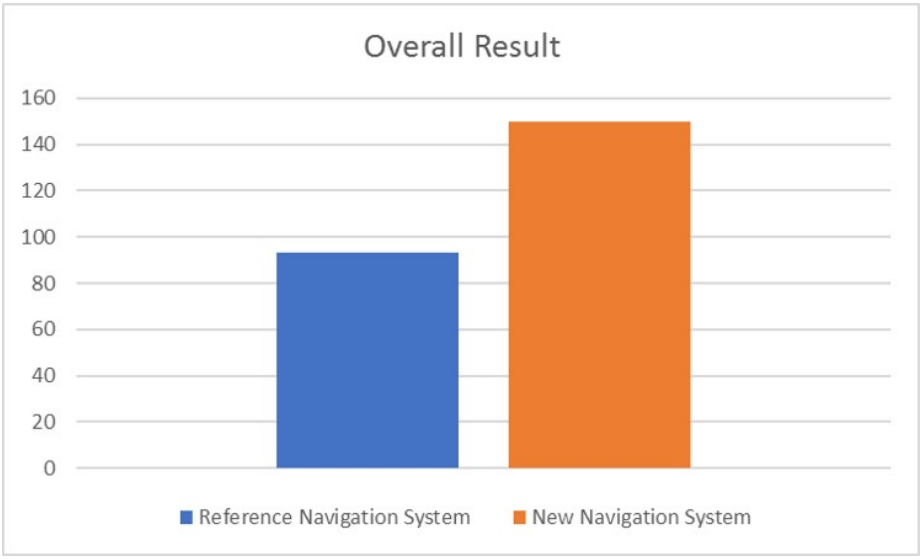


Figure 6.17: Likert-scale questions overall result.

The difference of both navigation systems for the individual questions is illustrated in Figure 6.18. To ensure the understanding of this figure, the questions and their labels according to the diagram are written down below:

1. The driving behavior of the robot is predictable.
2. The safety distances to obstacles are big enough.
3. The robot detects dynamic objects early enough.
4. The robot drives in my opinion the best way to get to its goal.
5. I feel safe near the robot.

According to Figure 6.18, probands agree more to the new introduced navigation system than to the reference navigation system. Due to the opinion of the probands, both systems are following the best path to reach the goal, but according to predictability, safety distance, obstacle detection and general safety feeling the new introduced navigation system has a higher rating than the reference system. Based on the two Figures 6.17 and 6.18, the probands in general do agree more to the new introduced system, than to the reference system.

To check whether the difference between the two systems is significant or not, a student-t test assuming different variances is applied to the data [65]. Based on a significant level of 5 percent (0.05) and a calculated p-value of 0.015532, the difference between the two navigation systems can be interpreted as significant. This outcome is mainly due to the fact, that the reference system is not really trying to avoid dynamic obstacles, its more about simply not crashing into dynamic obstacles.

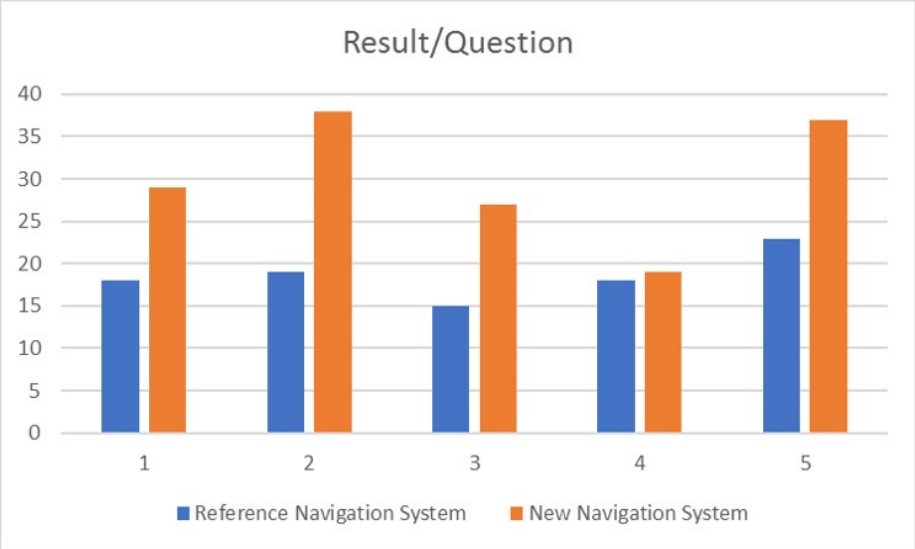


Figure 6.18: Individual results for the likert-scale questions.

Additionally to the likert-scale questions, the probands are also interviewed based on the open ended question »Describe your feelings and impressions regarding the driving behavior of the robot«. The answers of the probands can be summarized as following:

- **reference navigation system:**
  The system detects dynamic obstacles very late and is often not able to avoid them.

- **new navigation system:**
  The driving behavior of the robot is often curvy and jerky. It is not clear whether the robot recognizes the obstacle or not. Any kind of signal would be of benefit.

Due to the evaluation of the open ended question, the reference system is not really able to avoid dynamic obstacles but the driving behavior seems to be smooth. The new introduced navigation system on the other hand is definitely able to avoid dynamic obstacles and increases the general safety feelings, but the driving is more jerky.

Chapter 7

Conclusion

In this work we presented a concept to include dynamic obstacle avoidance in an already existing navigation system for a mobile robot platform.
Therefore, data delivered by a 3d sensor in form of a 3d point cloud is preprocessed to reduce noise and to extract points of interest. This step is followed by an obstacle detection step. Thus, obstacles are extracted from the filtered 3d point cloud using a clustering algorithm. After this extraction step, all obstacles are given to a multi-object tracking system, to estimate velocities and future positions of the obstacles. At the end, the information about position and velocity of dynamic obstacles is given to the navigation stack and based on the concept of velocity obstacles a avoidance maneuver is planned and executed.

The original idea was to detect and track dynamic obstacles in the environment based on the data delivered by a stereo vision system. As it is described in the Chapter Evaluation, the stereo vision system in combination with the provided hardware and platform cannot be used for real time applications. This has several reasons. On one hand, the range of the camera is limited by the resolution, while the resolution mode running the camera is limited by the computational power. Thus, the camera itself is able to provide data with higher resolution at a higher frequency, but the rest of the system is not able to process this huge amount of data fast enough. On the other hand, the camera data contains a lot of noise. This noise can be reduced by preprocessing steps, but these steps are again computational very expensive. To evaluate the overall system and to show that the integration of dynamic obstacle avoidance at the existing navigation system is possible, the data delivered by 3D laser range finder is used instead of the stereo vision camera.

Position as well as velocity information given by the tracker are quite accurate and good enough to tackle the goals of this thesis. Thus, a standard Kalman Filter based on a constant velocity model and the Hungarian algorithm to solve the data association problem are completely sufficient. A drawback of the tracking system is, that it is not able to track completely occluded obstacles.

The data delivered by the Kalman Filter is later on passed to an existing navigation stack. This navigation stack is extended by the concept of velocity obstacles. Velocity obstacles are outperforming the reference navigation system in special cases like crossings, but also have their disadvantages. In the

case where the robot is approaching the dynamic obstacle, the concept of velocity obstacles include these dynamic obstacles into the navigation system and avoiding it almost perfect. The reference system on the other hand can only avoid the obstacle, by stopping the robot and do a recalculation of the path. Using the new concept described in this thesis will cause some problems, when using it for real world scenarios. As it is described in the Chapter 6, the new system is not able to overtake the dynamic obstacle in a proper way, but this maneuvers has to be done very often in reality. During the evaluation also other problems came up as for example goal approaching. By definition, if a certain goal lies within a collision cone of a velocity object, the robot is not able to reach this goal, since all possible velocities the robot is able to reach within this collision cone are lethal. This problem can be only overcome with a high level controller or state machine and not with the basic approach of velocity obstacles.

Finally, an evaluation according to human factors is presented. This evaluation points out, that based on human feelings and opinions the new navigation system is given to the results of a student-t test significantly better than the reference system. Thus, the overall goal of the thesis has been achieved.

Chapter 8

Future Work

The performance of the new introduced navigation system is satisfying, but there might be several ways to possibly improve or extend this work.

First of all, the provided hardware setting is definitely not tuned for the purpose of computer vision in any aspect. Thus, testing the concept in combination with the stereo vision system in a new hardware setup may lead to improved results.

With the assumption that the data delivered by the sensor is accurate, also the tracking for the specific occlusion scenario can be probably improved by testing other approaches and do a better track management as for example multi-hypothesis tracking. Further also the motion model and the Kalman parameter can be chosen according to different types of obstacles as for example persons or robots to get an improved accuracy. Therefore, any kind of object detection algorithm has to be applied to the data.

In this thesis it was shown, that the concept of velocity obstacles is a pretty good approach to include dynamic obstacle avoidance in a navigation system. As it is already mentioned in the evaluation chapter, for the practical usage and realization, a high level controller as for example a state machine is needed for some special cases. This special cases have to be figured out and decisions procedures according to these cases have to be implemented.

List of Figures

List of Tables

Listings

List of Algorithms

Bibliography

[1] Christine M Anderson-Cook. Practical genetic algorithms. *Journal of the American Statistical Association*, 100(471):1099–1099, 2005.

[2] Robert L. Armstrong. The midpoint on a five-point likert-type scale. *Perceptual and Motor Skills*, 64(2):359–362, 1987.

[3] Christian Bailer, Kiran Varanasi, and Didier Stricker. Cnn-based patch matching for optical flow with thresholded hinge embedding loss. In *CVPR*. IEEE, 2017.

[4] Simon Baker and Iain Matthews. Equivalence and efficiency of image alignment algorithms. In *Proceedings of the 2001 IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 1090 – 1097, December 2001.

[5] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *Int. J. Comput. Vision*, 56(3):221–255, February 2004.

[6] D. H. Ballard. Readings in computer vision: Issues, problems, principles, and paradigms. chapter Generalizing the Hough Transform to Detect Arbitrary Shapes, pages 714–725. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[7] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *Int. J. Comput. Vision*, 12(1):43–77, February 1994.

[8] Marcelo Becker, Richard Hall, Sascha Kolski, Kristijan Maček, Roland Siegwart, and Björn Jensen. 2d laser-based probabilistic motion tracking in urban-like environments. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 31(2):83–96, 2009.

[9] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, 2016.

[10] Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, June 1991.

[11] Y. Y. Boykov and M. P. Jolly. Interactive graph cuts for optimal boundary amp; region segmentation of objects in n-d images. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 105–112 vol.1, 2001.

[12] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

[13] Maxime Derome, Aurelien Plyer, Martial Sanfourche, and Guy Le Besnerais. Moving object detection in real-time using stereo from a mobile platform. *Unmanned Systems*, 03(04):253–266, oct 2015.

[14] A. Dosovitskiy, P. Fischer, E. Ilg, P. Husser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, Dec 2015.

[15] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, January 1972.

[16] A. Ess, B. Leibe, K. Schindler, and L. van Gool. Moving obstacle detection in highly dynamic scenes. In *2009 IEEE International Conference on Robotics and Automation*, pages 56–63, May 2009.

[17] Andreas Ess, Konrad Schindler, Bastian Leibe, and Luc Van Gool. Object detection and tracking for autonomous navigation in dynamic environments. *The International Journal of Robotics Research*, 29(14):1707–1725, 2010.

[18] Loïc Fagot-Bouquet, Romaric Audigier, Yoann Dhome, and Frédéric Lerasle. Improving multi-frame data association with sparse representations for robust near-online multi-object tracking. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 774–790, Cham, 2016. Springer International Publishing.

[19] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, SCIA'03, pages 363–370, Berlin, Heidelberg, 2003. Springer-Verlag.

[20] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[21] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, March 1997.

[22] S.S. Ge and Y.J. Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 13(3):207–222, Nov 2002.

[23] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *Int. J. Rob. Res.*, 32(11):1231–1237, September 2013.

[24] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[25] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *ARTIFICAL INTELLIGENCE*, 17:185–203, 1981.

[26] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[27] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[28] H. Kieritz, S. Becker, W. Hbner, and M. Arens. Online multi-person tracking using integral channel features. In *2016 13th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 122–130, Aug 2016.

[29] Chanho Kim, Fuxin Li, Arridhana Ciptadi, and James M. Rehg. Multiple hypothesis tracking revisited. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 4696–4704, Washington, DC, USA, 2015. IEEE Computer Society.

[30] Pavlina Konstantinova, Alexander Udvarev, and Tzvetan Semerdjiev. A study of a target tracking algorithm using global nearest neighbor approach. In *Proceedings of the International Conference on Computer Systems and Technologies (CompSysTech03)*, pages 290–295, 2003.

[31] Till Kroeger, Radu Timofte, Dengxin Dai, and Luc Van Gool. Fast Optical Flow using Dense Inverse Search. In *ECCV*, 2016.

[32] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, mar 1955.

[33] L. Leal-Taix, G. Pons-Moll, and B. Rosenhahn. Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 120–127, Nov 2011.

[34] Laura Leal-Taix, Anton Milan, Konrad Schindler, Daniel Cremers, Ian Reid, and Stefan Roth. Tracking the trackers: An analysis of the state of the art in multiple object tracking. 04 2017.

[35] A. Leigh, J. Pineau, N. Olmedo, and H. Zhang. Person tracking and following with 2d laser scanners. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 726–733, May 2015.

[36] X. Li, K. Wang, W. Wang, and Y. Li. A multiple object tracking method using kalman filter. In *The 2010 IEEE International Conference on Information and Automation*, pages 1862–1866, June 2010.

[37] Matthias Luber, Kai O. Arras, Christian Plagemann, and Wolfram Burgard. Classifying dynamic objects: An unsupervised learning approach. *Autonomous Robots*, 26(2-3):141–151, 2009.

[38] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[39] Guan-Chun Luh and Wei-Wen Liu. An immunological approach to mobile robot reactive navigation. *Applied Soft Computing*, 8(1):30–45, 2008.

[40] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[41] S. Mansur, M. Habib, G. N. P. Pratama, A. I. Cahyadi, and I. Ardiyanto. Real time monocular visual odometry using optical flow: Study on navigation of quadrotors uav. In *2017 3rd International Conference on Science and Technology - Computer (ICST)*, pages 122–126, July 2017.

[42] Marta Marrón-Romera, Juan C. García, Miguel A. Sotelo, Daniel Pizarro, Manuel Mazo, José M. Cañas, Cristina Losada, and Álvaro Marcos. Stereo vision tracking of multiple objects in complex indoor environments. *Sensors*, 10(10):8865–8887, sep 2010.

[43] Hassan Mashad Nemati. Detection and tracking of people from laser range data. 2010.

[44] L. Matthies and A. Elfes. Integration of sonar and stereo range data using a grid-based representation. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 727–733 vol.2, April 1988.

[45] A. Milan, S. H. Rezatofighi, A. Dick, I. Reid, and K. Schindler. Online multi-target tracking using recurrent neural networks. In *AAAI*, February 2017.

[46] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[47] Juan Nieto, Jose Guivant, Eduardo Nebot, and Sebastian Thrun. Real time data association for fastslam. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 1, pages 412–418. IEEE, 2003.

[48] Jason M. O'Kane. *A Gentle Introduction to ROS*. CreateSpace Independent Publishing Platform, 10 2013.

[49] F. Oniga, E. Sarkozi, and S. Nedevschi. Fast obstacle detection using u-disparity maps with stereo vision. In *2015 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 203–207, Sept 2015.

[50] J. O'Rourke and N. Badler. Decomposition of three-dimensional objects into spheres. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(3):295–305, July 1979.

[51] A Pandey, S Pandey, and DR Parhi. Mobile robot navigation and obstacle avoidance techniques: A review. *Int Rob Auto J*, 2(3):00022, 2017.

[52] Cosmin D. Pantilie, Silviu Bota, István Haller, and Sergiu Nedevschi. Real-time obstacle detection using dense stereo vision and dense optical flow. *Proceedings of the 2010 IEEE 6th International Conference on Intelligent Computer Communication and Processing*, pages 191–196, 2010.

[53] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *CVPR 2011*, pages 1201–1208, June 2011.

[54] Aurlien Plyer, Guy Besnerais, and Frdric Champagnat. Massively parallel lucas kanade optical flow for real-time video processing applications. *J. Real-Time Image Process.*, 11(4):713–730, April 2016.

[55] Ciprian Pocol, Sergiu Nedevschi, and Marc-Michael Meinecke. Obstacle detection based on dense stereovision for urban acc systems. In *Proceedings of 5th International Workshop on Intelligent Transportation (WIT 2008)*, pages 18–19, 2008.

[56] Marc Pollefeys, Reinhard Koch, and Luc Van Gool. Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters. *International Journal of Computer Vision*, 32(1):7–25, Aug 1999.

[57] Long Qian, Xie Qiwei, Seiichi Mita, Kazuhisa Ishimaru, and Noriaki Shirai. Small object detection based on stereo vision. *International Journal of Automotive Engineering*, 7(1):9–14, 2016.

[58] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[59] Purushothaman Raja and Sivagurunathan Pugazhenthi. Optimal path planning of mobile robots: A review. *International Journal of Physical Sciences*, 7(9):1314–1320, 2012.

[60] S. H. Rezatofighi, A. Milan, Z. Zhang, Q. Shi, A. Dick, and I. Reid. Joint probabilistic data association revisited. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3047–3055, Dec 2015.

[61] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut": Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, August 2004.

[62] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl. In *In Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE.

[63] Amir R Soltani, Hissam Tawfik, John Yannis Goulermas, and Terrence Fernando. Path planning in construction sites: performance evaluation of the dijkstra, a, and ga search algorithms. *Advanced engineering informatics*, 16(4):291–303, 2002.

[64] Luciano Spinello, Matthias Luber, and Kai O. Arras. Tracking people in 3D using a bottom-up top-down detector. In *Proc. of the Int. Conf. on Robotics & Automation (ICRA)*, Shanghai, China, 2011.

[65] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908.

[66] P. Vadakkepat, Tong Heng Lee, and Liu Xin. Application of evolutionary artificial potential field in robot soccer system. In *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference (Cat. No. 01TH8569)*, pages 2781–2785 vol.5, July 2001.

[67] B. Wang, S. A. R. Florez, and V. Frmont. Multiple obstacle detection and tracking using stereo vision: Application and analysis. In *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 1074–1079, Dec 2014.

[68] B. Wang and V. Frmont. Fast road detection from color images. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 1209–1214, June 2013.

[69] Andreas Wedel, Annemarie Meißner, Clemens Rabe, Uwe Franke, and Daniel Cremers. Detection and segmentation of independently moving objects from dense scene flow. In Daniel Cremers, Yuri Boykov, Andrew Blake, and Frank R. Schmidt, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 14–27, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[70] Martyna Weigl, B Siemiatkowska, Krzysztof A Sikorski, and Andrzej Borkowski. Grid-based mapping for autonomous mobile robot. *Robotics and Autonomous Systems*, 11(1):13–21, 1993.

[71] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. In *2013 IEEE International Conference on Computer Vision*, pages 1385–1392, Dec 2013.

[72] Gang Xu and Zhengyou Zhang. *Epipolar Geometry in Stereo, Motion, and Object Recognition: A Unified Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[73] Min Zhang, Peizhi Liu, Xiaochuan Zhao, Xinxin Zhao, and Yuan Zhang. An obstacle detection algorithm based on u-v disparity map analysis. In *2010 IEEE International Conference on Information Theory and Information Security*, pages 763–766, Dec 2010.

[74] Zhengyou Zhang. Determining the epipolar geometry and its uncertainty: A review. volume 27, pages 161–195. Kluwer Academic Publishers, March 1998.

[75] Dingfu Zhou, Vincent Frémont, Benjamin Quost, Yuchao Dai, and Hongdong Li. Moving object detection and segmentation in urban environments from a moving platform. *Image and Vision Computing*, 68:76–87, dec 2017.