Hochschule Aalen

Entwicklung von Navigationssoftware für mobile Robotersysteme und Simulation

by

Tristan Schwörer

Matriculation number: 71336

A bachelor thesis submitted in partial fulfillment of the requirements for the degree of the

Bachelor of Engineering (B. Eng.)
in Mechatronics
at Aalen University

Supervisor:
Prof. Dr. Stefan Hörmann (Aalen University)

Submitted on:
April 13th, 2021

Preface

This bachelor thesis is part of the bachelor program in mechatronics at Aalen University and is supposed to take place in the 7th Semester. It covers the theoretical and practical work between November 2nd 2020 and April 13th 2021.

This work took place at the laboratory for mobile roboic systems of the faculty optics and mechatronics at aalen university and was completely independent of any other party and company.

Diese Bachelorarbeit ist fester Bestandteil des Bachelorprogramms Mechatronik an der Hochschule Aalen und soll im siebten Semester statt finden. Die hier dokumentierte Arbeit wurde zwischen dem 2. November 2020 und dem 13. April 2021 realisiert.

Die praktische sowie die theoretische Arbeit fand im Labor für mobile Robotersysteme der Fakultät Optik und Mechatronik an der Hochschule Aalen statt und ist komplett unabhängig von jeglicher anderen dritten Partei oder Firma.

Abstract

This bachelor thesis is about the concept, setup/development and testing of a software stack used for the autonomous navigation in an environment defined by the rules of the carolo cup.

The aim of this stack is lane following and obstacle avoidance based on a sensor data of the environment. This thesis extends the work of Prof. Hörmann who provided the road detection and is supposed to be used by the carolo cup team of university aalen in the future. Even though the robot used in this thesis does not satisfy the rules of the carolo cup the stack should be configurable for different robots aswell.

The robot is equipped with a lidar, a camera, wheel encoders and an imu. The data of these sensors will be filtered and processed using existent ros packages as well as newly developed ones. The resulting data will be fed into the navigation stack that then determines the best route for the robot.

Since there wasn't a driving robot available at the start of this thesis the task of simulating the robot with all of its sensor data and actors has been incorporated into the subject of this work.

Kurzfassung

Diese Bachelor-Thesis handelt von der Erstellung eines Konzepts, dem Aufbau und der Entwicklung eines “Software-Stack” und dessen Testens für die autonome navigation in einer, durch das Regelwerk des carolo cups beschriebenen, Umgebung.

Das Ziel dieses “Software-Stacks” ist, der Spur einer Straße zu folgen und dabei potentiellen Hindernissen auf der Straße auszuweichen. Diese Thesis führt die Arbeit von Prof. Hörmann der die von ihm Entwickelte Spurerkennung zur Verfügung stellte und soll in der Zukunft vom Carolo-Cup Team der Hochschule Aalen verwendet werden können. Obwohl der in dieser Arbeit verwendete Roboter nicht konform zum Regelwerk des Carolo-Cups ist, soll der Stack auch für andere Roboter konfigurierbar sein.

Der Roboter verfügt über einen Lidar, eine Kamera, Rad-Encoder und einen IMU (inertia measurement unit). Die Daten dieser Sensoren werden gefiltert und dann mit bestehenden ros packages und selbst entwickelten aufbereitet. Die resultierenden Daten werden dann an den Navigation Stack übergeben, der dann die beste Rute ermittelt.

Da zu Beginn dieser Arbeit kein vollständig funktionierender Roboter verfügbar war wurde das Teilthema der Simulation des Roboters mitsamt aller seiner Sensoren und Aktoren in das Thema der Thesis aufgenommen.

Acknowledgement

At this point I would like to thank the following people for supporting me during my bachelor thesis:

- **Prof. Dr. Stefan Hörmann** for being my supervisor during this time and for allways helping me with new ideas and approaches.
- **Prof. Dr. Arif Kazi** for being the second supervisor and helping me with ideas regarding the structure of the thesis.

Extending to that i would like to thank my Family and Friends that supported me during this period.

Table of Contents

1. Theoretical Background

This chapter will cover the needed theoretical background about the Gazebo Simulation, the Sensor Plugins, ROS and all of the used ROS packages.

1.1. ROS

1.1.1. Nodes

1.1.2. Plugins

1.1.3. Topics/Services/Actions

All three are possibilities for the data exchange between nodes.
According to ::::: Services and actions can be used like the subscriber/publisher structure but are meant for the intercommunication between nodes. A service is more or less a function in a different node that has the option to receive data and respond to it.

1.1.4. RVIZ

1.1.5. REP

REP's (short for ROS enhanced proposals) are guidelines made and maintained by the ros community. It is highly advisable to follow the guidelines as much as possible.
Complying to these guidelines allows external people easier comprehension of the structure of the robot and eliminates misunderstandings.
The most important REP's in this project are REP 103 and REP 105.

REP 103

"This REP provides a reference for the units and coordinate conventions used within ROS"[1]

Coordinate Frame

- **X-Axis** - Forward
- **Y-Axis** - Left
- **Z-Axis** - Up

Units
Units will always be represented in SI Units and their derived units.

The order of preference for rotations

1. Quaternion
2. Rotation matrix
3. fixed axis roll, pitch, yaw
4. Euler angles

[1]

REP 105

"This REP specifies naming conventions and semantic meaning for coordinate frames of mobile platforms used with ROS."[2]

REP103 Applies for all fixed coordinate frames.
Coordinate Frames

- **base_link** is a fixed frame on the robot base. It serves as the reference points for all of hardware mounted on the robot itself like sensors.
- **odom** is a world fixed frame that serves as the reference for the pose of the robot. Since the pose of the robot will drift over time it wont serve as a good long term reference.
  In most cases the odom frame will be computed using localization sensors like wheel odometry, imu's, visual odometry, etc. which leads to a continuous frame.
- **map** is a world fixed coordinate frame that serves as the reference for the odometry frame. It is also the base for a map of the environment such as the ones provided by slam algorithms. The frame is time discrete since it is mostly computed by localization algorithms.

That tree can be extended by an earth frame that would be the reference for the localization of the map in the earth. Which is useful, for long range robot platforms.[2]

1.1.6. TF

In most cases robots that are controlled by ros have a so called tf_tree. This tree is the coordinate frame structure of the robot. In it every sensor and actor has its own coordinate frame.
The structure in most trees of mobile platforms is quite similar which is caused by the REP105 (ROS Enhanced Proposals) this contains a definition of recommended names for the robot frames and their order in the tree. But it should be noted that not every

frame that is defined in the norm has to be in every tree. The basic structure mostly starts at a so called fixed frame. This Frame will be the not changing frame in the environment. At moving robots this is often earth, map or odom, while in stationary robots this can even be base_link.

The tree is normally build up like in the following image.
TF2 is the successor of TF and is a very powerful tool in the ROS environment. With it it is possible to transform sensor_msgs and geometry_msgs from one frame in another. Furthermore it offers the possibility to transform old data into the present or at any other point in the past.

URDF and xacro

The robot hardware description consists of one or more URDF(Unified Robot Description Format) based xml file. Its purpose is to define the shape and geometric of every part of the robot.

robot_state_publisher

This package uses the robot hardware description and builds up the tf_tree using static_transform_publishers.

1.2. Gazebo

1.2.1. Plugins

Gazebo offers a wide selection of pre made plugins that can be incorporated into a simulated robot by attaching the plugin to the right tf_frame and configuring its parameters.

Camera

Lidar

Differential Controller

IMU

1.2.2. Models

1.3. navigation stack

1.3.1. mobe_base

1.3.2. global_planner

base_global_planner

1.3.3. local_planner

teb_local_planner

base_local_planner

dwa_local_planner

1.3.4. costmap_2d

global map

local map

layer

1.4. cartographer

1.5. Carolo-Cup

The carolo cup is an event hosted by the University Braunschweig and is an event in which the teams of many different universities can compete against each other and present their work and progress in the field of autonomous driving.
There are two different levels of difficulty the carolo basic cup and the carolo master cup.

2. Limitations and Requirements

Before diving into the details and developing concepts the guidelines and requirements of the project has to be defined.
This thesis aims for a navigation of a robot in an environment that is similar to the carolo cup but deviates at some parts.

2.1. Robot and Environment

While the robot has very strict regulations in the carolo cup theses will not apply here.

At the time of this thesis there is no driving real robot available. That is the reason, to use a simulated environment.
The robot that will be used is a differential drive robot from the company Parallax with a diameter of 450mm.

Equally to the carolo cup regulations the lane width is defined by double robot width and will be set to 900mm.
The Robot will be equiped with the following sensors:

- Lidar
- Wheel encoder
- IMU
- Camera

Additional it will feature a motor driver for differential drive steering.

2.2. Software

Generally the software will be developed for ROS-Noetic.
The programming language will be mostly C++ to allow uniformity in the software stack.
Like in the carolo cup the software is not supposed to have any connection to systems outside of the robot.

2.3. Simulation

Since the environment will be simulated the Simulator has to have the following features.

- Sensor plugins with configurable error and ROS interfaces
- Differential drive plugin
- custom models integration
- URDF conversion
- Not too computationally heavy

The simulation will mostly focus on sensor data. That is why sensor plugins with configurable error and a ROS interfaces are needed. Like this the data will be as similar as possible to the real world.

In addition to the sensor plugins the simulator needs to provide a plugin for differential drive steering. This will be the replacement for the motor controller of the real robot.

Custom models is a strict requirement since this thesis focuses on a very specific robot. Furthermore the integration of custom models is necessary to put the robot in different road scenarios.

A URDF conversion plugin is very important like this differences between the simulated robot and the tf-tree in ROS can be avoided and the robot will be defined in one file only.

To get the best correlation between simulation and real world the simulator should be able to run as close to real time as possible. This will make the simulated sensor data way more reliable and puts the nodes of the navigation_stack under the right load.

2.4. Navigation

The navigation is supposed to cover free driving with out obstacles, as well as with static obstacles avoidance. It will not cover dynamic obstacles, road sign detection or driving situations like intersections and parking.

Development of an entire stack exceeds the content of this thesis, so an open source navigation project will be used which needs to satisfy the following requirements.

- Sensor input.
- Goal pose input
- 2D mobile platform support using the conventional drive systems like Ackermann and differential

- Path planning in respect to the robots kinematic and shape, as well as the environment detected by the sensors.
- Path planning and navigation in totally unknown environments
- Velocity output as linear and angular velocities

3. Concept

The selection of navigation stacks is quite sparse. Especially considering the defined requirements. In general when it comes to navigation there are only two well documented stacks to choose from.

- navigation_stack
- MoveIt

In contrast to the navigation_stack MoveIt is largely used for the path planning and navigation of industrial robot arms and therefore not suited for this application.
The navigation_stack provides a general setup proposal which seems to be a good starting point for robot navigation, but it has multiple parts that will be modified to adhere to the defined requirements, that will be discussed in the following part.

3.1. platform specific nodes

Since the platform specific nodes are unique to each robot these will need to be adjusted.

3.1.1. sensor sources

The following sensors will be predefined for this concept:

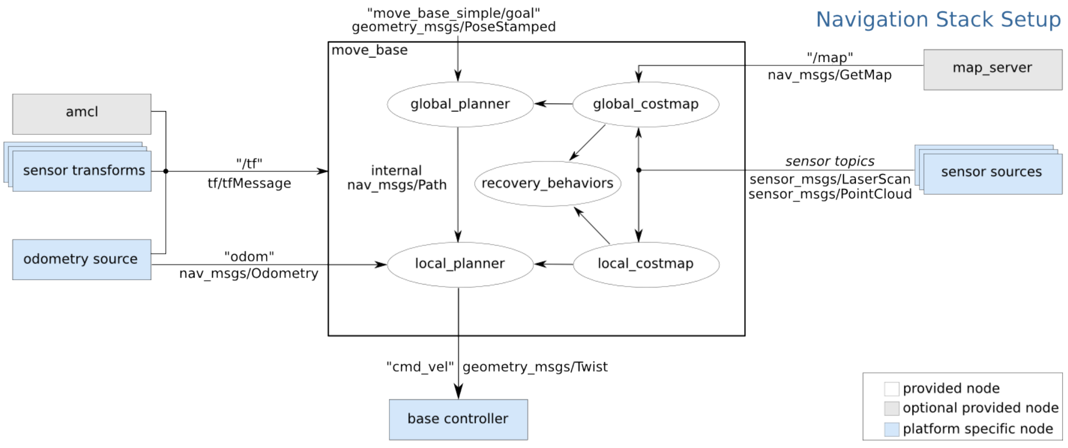


Figure 3.1.: navigation stack setup

- lidar
- wheel encoder
- imu
- camera

Since these sensors will certainly have noise we firstly need to add additional filters for the sensor signals.

3.1.2. Odometry source

The odometry is all ways a result of sensor data. Therefore we can make a direct connection between the sensor filters and the odometry input. These Filters will also feature nodes that transform the incoming sensor data into a useable format.

3.1.3. Sensor transforms

To know the position of every sensor relative to the robot and his odometry the tf_tree has to be build. While this can be realised using static transform publishers there is also a cleaner way using the ros package robot_state_publisher. It then requires a robot description in URDF format that specifies the relations between everything mounted on the robot. The transformation between the base of the robot and the odom will be build by the filtered odometry.
The remaining platform specific nodes are all available since the simulated robot will be used and provides a motor controller, as well as all of the sensors and data sources.

3.2. Optional nodes

The Map is a representation of the robots environment. If this is known from the start it is known, where the robot can go and where not.
In this scenario the robot will always start blind, meaning it will not know anything about its surrounding other than that it can expect a road to be there, which makes the map_server of the navigation_stack and its functionality to convert prerecorded maps redundant.
Adding to that amcl (the localization package of the navigation_stack) will no longer work since it tries to find a position in a predefined map based on the current sensor signals.
Knowing the environment and the position of the robot in it is an important part in robot navigation since it allows to send goals that are relative to the environment and not to the robot. That is why the usage of a SLAM algorithm becomes highly useful. This node supplies both the current map and the position of the robot in it.
This node will publish the transform between the map and the odom frame so the

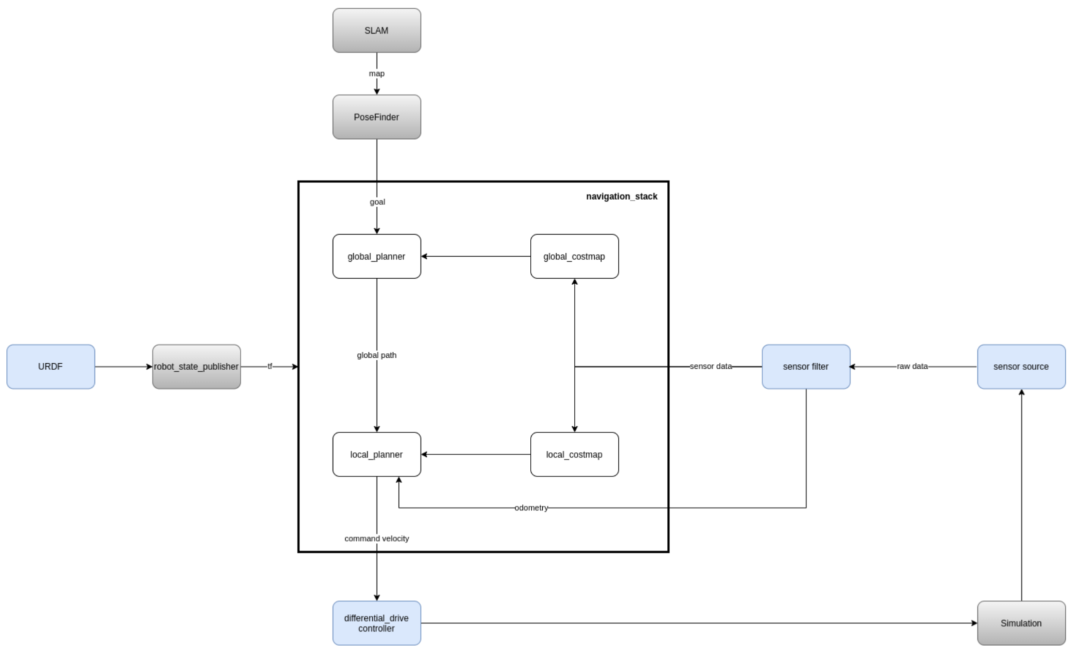


Figure 3.2.: Updated navigation concept

position of the robot and every sensor signal can be determined relative to the map frame.

3.3. Provided nodes

The provided nodes do not have to be reordered but the recovery behaviors will be removed. These will be incorporated in the incoming goals instead.

3.4. PoseFinder

The job of this node is to extract the pose of a goal from the sensor data or the map (if available).

The requirements of this node will be defined in the ”Configuration and testing” section

3.5. Resulting concept

The following simplified schematic is the concept of the navigation stack setup. Not all of the connections between the nodes will be highlighted, to make the schematic easier to understand.

The PoseFinder will first find a goal based on the current state of the map and the sensor data and sends it to the navigation stack. This then uses the filtered sensor data to determine, where the robot is, where it is allowed to go and where not. The cascading planners then determine first a rough, collision free, path and then a path

that is possible for the kinematics of the robot. This path will be send to the motor controller of the simulated robot.
This procedure will be repeated at a configurable frequency so the robot will never reach its finish. This is necessary since it is unknown if the goal, in the space that has not been explored yet, is perfectly on the future road, or if it is in an obstacle.

4. Selection

This chapter will cover the package selection and the setup of all nodes in the concept. It will also give attention to the configuration of the individual nodes.
The selection process will mainly focus on the defined requirements.

4.1. Simulation

There are many options, when it comes to robot simulation, which makes a selection mandatory. The chosen Simulator then needs to be configured and equipped with models, sensors and a drive system.

4.1.1. Selection

To begin of the selection process a group of reasonable simulators needs to be collected. The two selected options are Gazebo and V-REP since they are the two most used robotics 3D simulators [3].

Both simulators seem to full fill most of the defined requirements to a certain extend, while Gazebo seems to have a easier installation process and integration into ROS since it is included in the default packages of ROS noetic [4] since it developed by the Open Source Robotics Foundation as the default simulator for ROS.
As well as Gazebo V-REP offers plugins and URDF conversion for custom models but an even bigger selection of mobile robot models. Which unfortunately can only be used as examples since the required models are very specific.
In contrast to V-REP, Gazebo does not contain an integrated model editor.
Based on computational load compared in the Paper of L. Pitonakova et al[5] and the setup differences between both simulators, gazebo will be chosen for this project. But both simulators would have been an excellent choice.

4.1.2. Model

Since Gazebo doesn't have an integrated model editor the freeware blender will be used for the model generation of the environment.
As described in the requirements the robot models will be generated using URDF which will be covered later.

4.2. Navigation

As described in the Concept part the ros navigation stack will be used in this project. The selection of all of its parts will be discussed in the following sections

4.2.1. Planners

Since the planners will consist of two parts (the global planner and the local planner) the nodes of these have to be selected individually.

global planner

For the global planner the following two options provided in the navigation stack will be considered

- base_global_planner
- navfn

Here the choice is fairly easy, since base_global_planner is the successor of navfn. It still supports the the behavior of navfn, but it offers more options, such as A* planning algorithm instead of Dijkstra.

A* and Dijkstra explena-tion in theoretical

local planner

In contrast to the global planner there are way more options for the local planner node. Like the global planners the local planners will be selected using the options offered in the description of the nav_core.

- base_local_planner
- dwa_local_planner
- eband_local_planner
- teb_local_planner
- mpc_local_planner

[6]
To choose the local planner the requirements have to be defined first. Since the global planner is taking care of the obstacles and the roadlanes the local planner has the general task of following the global path and creating a command velocity that is feasible in regards to the kinematics of the robot.

Smooth lane changes are highly wanted in this project. this will help the camera and therefore the road detection to keep seeing the road during swapping the lanes.

A time constraint could be used to prevent oscillations around the global path while trying to follow it as close as possible.
As described in its documentation teb_local_planner optimizes the trajectory with respect to trajectory execution time, separation from obstacles and compliance with kinodynamic constraints in the runtime. site

4.2.2. costmap

For the costmap the highly configurable costmap_2d package will be used. This package is the default package of the navigation_stack and has no competing packages.

global costmap

local costmap

dynamic_cost_layer

Enhancing to the plugins provided in the navigation stack this layer handles inflation of cells with a configurable cost decay and radius.

This behaviour can be used to inflate the left road marking in the global costmap to force the global plan on the right side of the road. The plugin can also be used to inflate cells with zero cost, which is useful to guarantee a free right lane or to give some free space around obstacles located on the road.

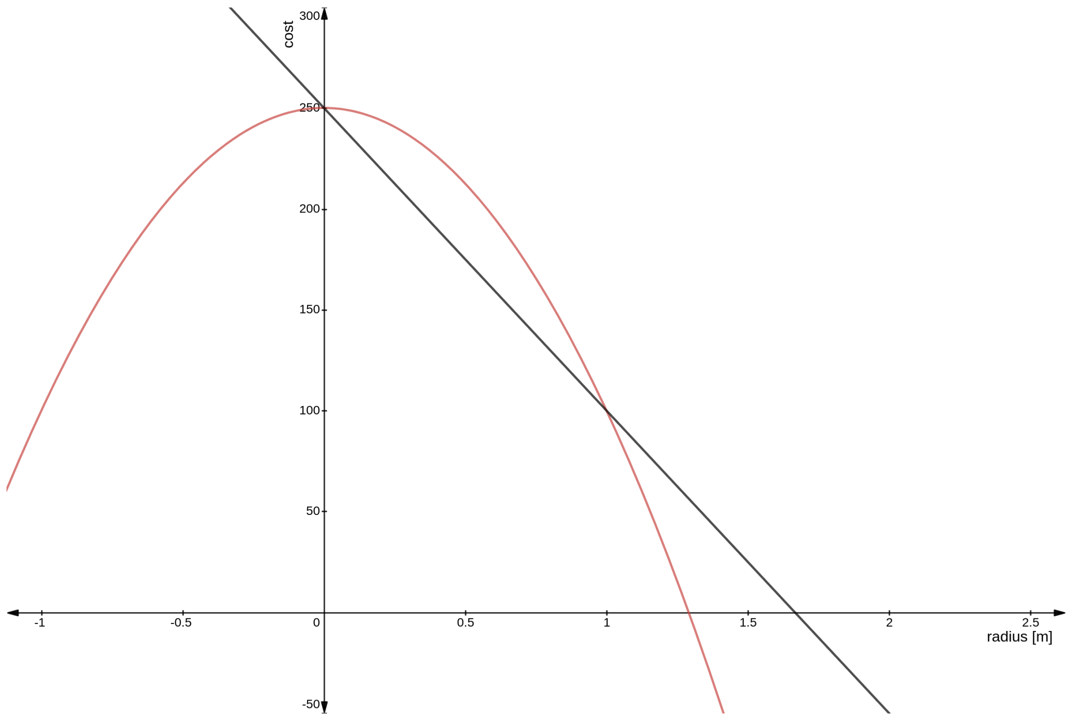
The layer receives a message of type sensor_msgs::PointCloud on a configurable topic. This PointCloud is expected to feature Channel Values for the inflation radius, the maximal and the minimal cost for each individual point.

Since we can't assume that the incoming points will be in the frame of the costmap the points in the costmap have to be transformed into the right frame using tf2.

To minimize the computation load a bresenham based algorithm for the circle rasterization will be used. Now the point symmetry around the cell can be used to further minimize the computational load and only $\frac{1}{8}$th of the circle has to be computed. The rasterization process can be described by the following image.

Adding to the typical behavior of the bresenham rasterization the area of the circle will be filled with the cost specified for the point. For this a linear decaying 1$^{\text{st}}$ degree function will be used which requires the computation of the distance of the cell to the center of the circle.
Since this will still require the usage of a square root for each cell in the circle This will be optimized as well.

The goal here is to use a function that contains only the squared distance, which still represents a decaying trend. This requirement rules out every function with an odd degree, as well as all functions with an x offset. This leaves all functions with an even degree from which we choose the 2nd degree function to reduce square operators. The comparison between the two functions can be seen in the picture below.



4.3. Odometry

4.3.1. Encoder

4.3.2. IMU

4.3.3. Improvement using SLAM

4.4. Laser_Filter

4.5. PoseFinder

This Node is part of the arlo_navigation package. Its job is, to find the pose of the next possible goal and send it to the move_base action client. Based on the data we gather from the observing sensors the only two sources to gather data about the road situation in front of the robot is the extracted camera data and the data from the last time the robot was in the same area than can be found in the map generated by SLAM. Since we don't have the map in the first round of the robot gathering goals from the live camera data is more important.

4.5.1. Using current camera data

The easiest way to get new goals would be to take the last point of the polynomial provided by the road detection as the position and calculate the yaw angle of the new goal with the gradient of the last two.
While this is a logical approach in an ideal scenario, it certainly will not work in a realistic one since we can't assure a continuous data stream from the road detection. This is mostly caused by the camera not always seeing enough of the road which can for example be caused by the following reasons:

- obstacle covering the markings
- while driving a corner the camera isn't always pointed tangential to the curvature
- steering during obstacle avoidance
- noise in camera data

This suggest that a prediction for the possible upcoming road is needed. To a small extend this is provided by the polynomials of the road detection, but the have a restricted domain, after which the error will rapidly increase.

4.5.2. Approximations

Since the road mostly consists of circles of varying radii and origins, it is self evident, that using the polynomials in their restricted domain to represent a section of a circle will give a better estimate.
This circle can be calculated using the following linear least-square approach:
In theory this approximation should work for almost straight sections as well, but the radius and the origin will trend to infinity and caused by the camera noise and the inaccuracy of the road detection the representation of the road will get worse and worse.
This leads to the following linear least square approach for straight lines:
Switching between the result of the two approximations will result in the best result for both scenarios. This switching will be triggered by the radii of the approximated circles exceeding a configurable threshold.
The next step is a goal extraction from the chosen approximation. This goal will be calculated at a given distance from the robot origin on the approximated route. In contrast to the approximated lines, the circles have a defined angle of trustworthiness. Otherwise small circles would cause the goal to be way of the prediction. The orientation of the goal will then be determined using the differentiation of the function. With the calculated points on either both circles or on both lines the mean can be calculated and represents the new goal for the robot.

4.5.3. goal from map

If the robot is running a SLAM algorithm during the first round the map should be finished once the robot passes its start point. Then the approximation is unnecessary since extracting goals directly from the slam map is more efficient and provides goals that will always be on the road. Furthermore the distance, at which goals will be found can be increased, which results in higher speeds and/or lower planner frequencies.
To find a goal in the SLAM map a circle rasterization algorithm will be used. This algorithm finds every cell on a circular path around the robot and its associated value in the map. The values outside of a given FOV (field of view) can be eliminated. Then the remaining values with a larger likelihood than a configurable threshold will be reduced to one point by taking the mean value of all of them.
The orientation of the goal is then determined by using the approximation algorithms.

4.6. MarkFreeSpace

The purpose of this node is to provide data to the SLAM algorithm as well as to the costmaps. This data consists from the points on the polynomials of the road detection in combination with the filtered points of the laser scan.
For SLAM and the obstacle layer of the costmap the node simply transforms the data into the same frame and casts it into the form of a sensor_msgs::PointCloud2.
The data of the dynamic cost layer has to contain more information. It is in the form of a sensor_msgs::PointCloud and contains channel values for point individual inflation radius, min-cost and max-cost. By giving the Layer point individual inflation parameters the node provides information about the cells on the left road marking that need to have inflated cost values and information about the points on the right road marking that should have a clean zone around them. Furthermore the node provides points of obstacles that are on the road and clears the inflated cost around it. Which will allow the global planner to make a plan for passing the obstacle.

4.7. SLAM

5. Results and Discussion

6. Conclusion

6.1. Personal conclusion

7. Outlook

8. List of Figures

9. List of Tables

10. References

[1] Mike Purvis Tully Foote. *Standard Units of Measure and Coordinate Conventions.* `https://www.ros.org/reps/rep-0103.html`. Accessed: 2021-02-20. Oct. 7, 2010.

[2] Wim Meeussen. *Coordinate Frames for Mobile Platforms.* `https://www.ros.org/reps/rep-0105.html`. Accessed: 2021-02-20. Oct. 27, 2010.

[3] M. Santos Pessoa de Melo et al. "Analysis and Comparison of Robotics 3D Simulators". In: *2019 21st Symposium on Virtual and Augmented Reality (SVR)*. 2019, pp. 242–251. DOI: `10.1109/SVR.2019.00049`.

[4] •. *ROS Index.* `https://index.ros.org/packages/#noetic`. Accessed: 2021-02-26. Feb. 26, 2021.

[5] Lenka Pitonakova et al. "Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators". In: (2018).

[6] *nav_core documentation.* `http://wiki.ros.org/nav_core?distro=noetic`. Accessed: 2021-03-01.

Appendix

A. Additional Topics

B. Source Code

Eidesstattliche Erklärung

Name: Schwörer	**Vorname:** Tristan
Matrikel-Nr.: 71336	**Studiengang:** Mechatronik

Hiermit versichere ich, **Tristan Schwörer**, an Eides statt, dass ich die vorliegende Bachelorarbeit
an der **Hochschule Aalen**
mit dem Titel **„Entwicklung von Navigationssoftware für mobile Robotersysteme und Simulation“**
selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht.
Ich habe die Bedeutung der eidesstattlichen Versicherung und prüfungsrechtlichen Folgen (§23 Abs. 3 des allg. Teils der Bachelor-SPO der Hochschule Aalen) sowie die strafrechtlichen Folgen (siehe unten) einer unrichtigen oder unvollständigen eidesstattlichen Versicherung zur Kenntnis genommen.

Auszug aus dem Strafgesetzbuch (StGB)

§156 StGB Falsche Versicherung an Eides Statt Wer von einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

Ort, Datum	Unterschrift