Small-Scale Autonomous Racing using only LIDAR

Vincent Wölfer, Sarthak Langde, Rajat Sahore and Karel Smejkal
Technische Universität Berlin, DAI-Lab,
February 16, 2020

***Abstract*—This paper presents the algorithms and system architecture of an autonomous race car which first explores an unknown track and utilizes this data to drive the track as fast as possible. The proposed solution combines state of the art techniques from different fields of robotics which are implemented in the *Robot Operating System* (ROS) [1]. Specifically, object detection using only LIDAR, pose estimation and motion control are incorporated into one autonomous race car. This robotic system was tested in an artificial environment at the Technische Universität Berlin. We discuss the implementation approach and decisions that were used in development of the car and present an experimental evaluation of our solution.**

***Index Terms*—autonomous, car, racing, lidar, ros**

I. Introduction

Although the first experiments with autonomous cars dates back to the 1980 [2], the improvements in recent years, in both hardware and software technology, allowed for real application and deployment of autonomous cars on the roads. Every year more than 1.2 million people are killed in traffic and it is leading cause of deaths of young people in many countries. Not only this, but also things, such as spending hours in traffic jams and better parking decisions, is what drives the research on autonomous vehicles [3]. The goal of self-driving vehicles is to improve safety, convenience and also reduce expenses compared to the traditional vehicles. According to the National Highway Traffic Safety Administration there are 5 levels of autonomy [4]:

- **Level 0 - No-Automation.** The driver is in charge of the primary vehicle controls at all times, and is he responsible for monitoring the roadway and for safe operation of all vehicle controls. Examples include systems that provide only warnings as well as systems providing automated secondary controls such as wipers, headlights, turn signals, hazard lights and more [4].
- **Level 1 – Function-specific Automation**: Automation at this level involves one or more specific control functions; if multiple functions are automated, they operate independently from each other. Examples of function-specific automation systems include: cruise control, automatic braking, and lane keeping [4].
- **Level 2 - Combined Function Automation**: This level involves automation of at least two primary control functions designed to work in unison to relieve the driver of control of those functions. The driver is still responsible for monitoring the roadway and safe operation and is expected to be available for control at all times and on short notice [4].
- **Level 3 - Limited Self-Driving Automation**: Vehicles at this level of automation enable the driver to cede full control of all safety-critical functions under certain traffic or environmental conditions and in those conditions to rely heavily on the vehicle to monitor for changes in those conditions requiring transition back to driver control. An example would be an automated or self-driving car that can determine when the system is no longer able to support automation [4].
- **Level 4 - Full Self-Driving Automation (Level 4)**: The vehicle is designed to perform all safety-critical driving functions and monitor roadway conditions for an entire trip. Such a design anticipates that the driver will provide destination or navigation input, but is not expected to be available for control at any time during the trip. By design, safe operation rests solely on the automated vehicle system [4].

A. *Problem Statement*

Goal of our project was to construct a level 4 - full self-driving small-scale four-wheel drive vehicle which can race on an previously unknown track which is autonomously explored and mapped during the first round after which the vehicle starts to drive as fast as possible in the then explored track. For localization and mapping a SLAM algorithm is used which only uses wheel odometry and lidar data and thus is not relying on vision systems such as cameras. The track is marked by traffic cones, which are detected using the lidar data and then used to infer the track boundaries.

B. *Overview*

This brief introduction is followed by chapter II which goes into the details about the system design and architecture as well as the chosen algorithms to tackle problems such as mapping, path planning and track detection. The next chapter III explains implementation details of the algorithms and approaches described in II which then leads to chapter IV where those techniques are evaluated and possible limitations are explored. We then conclude our research in chapter V and give an outlook for future research.

II. System Design and Software Architecture

To control and ensure communication between processes and different sensors of the small scale car, our solution uses the Robotic Operating System (ROS) - an open-source, meta-operating system. ROS is a middleware framework build to operate on an already existing operating system. This distributed framework of processes (called Nodes) allows us to control all sensors available in the car including lidar, front RGBD, back RGB, Inertial Measurement Unit (accelerometer + gyroscope) and battery. There are several advantages of using ROS inluding [1]

- **Scaling** ROS is suitable solution for large runtime systems and for large development processes.
- **Language independence**: ROS offers implementation in Python, C++, and Lisp, and also experimental libraries in Java and Lua.
- **Open-sourced ROS packages**: Already developed open-source ROS packages for popular robotics algorithms.

The hardware construction and components of the robot used are shown in Fig. 1. As mentioned in the problem statement, our solution does not rely on any camera data, therefore the front and back cameras are turned off. In the following subsections we will look closely at the design and algorithms we chose as the most suitable solution for the race car.

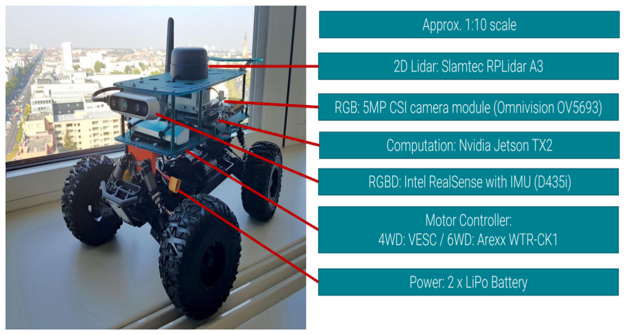
Approx. 1:10 scale
2D Lidar: Slamtec RPLidar A3
RGB: 5MP CSI camera module (Omnivision OV5693)
Computation: Nvidia Jetson TX2
RGBD: Intel RealSense with IMU (D435i)
Motor Controller: 4WD: VESC / 6WD: Arexx WTR-CK1
Power: 2 x LiPo Battery

Fig. 1. Hardware components of the robot [5]

A. *Simultaneous Localization and Mapping*

In order for the robot to map its surrounding area and localize itself in it SLAM was used. SLAM stands for *Simultaneous Localization and Mapping* and was first proposed by Durrant-Whyte et al. [6] to solve the chicken-and-egg problem of an autonomous agent localizing itself in an unknown environment while also mapping said environment.

For our project SLAM was mainly used to get an accurate estimation of the robots position which is necessary for path and motion planning. Furthermore, as SLAM estimates both the current robots pose as well as the environment, the SLAM algorithm was required to perform well in order to accurately record lidar data which is then leveraged for the object detection further described in II-B1.

1) Algorithm Choice: While SLAM was definitely required as an important component for our project it was not our focus of development. Therefore, we choose to use an already existing ros package which implements a SLAM algorithm. This enables us to focus on other components and gives us a well designed and tested implementation. To evaluate our possibilities we used the comparison of existing SLAM implementations in ROS by Santos et al. [7]. With this evaluation and some brief testing we choose the *slam-karto* package [8] which is based on *open karto* [9].

Among other benefits we choose *slam-karto* because of its Loop Closure Detection capabilities. Although modern SLAM algorithms implement sophisticated techniques to minimize errors, these can not be fully eliminated and are adding up over time. To further reduce these errors Loop Closure Detection can be used. When the robots recognizes a previously visited place it is able to accurately determine its position and the error which accumulated over time while driving around. This error can then be corrected in retrospective because the exact position before and after the error occurred is now known. Since our scenario consists of a circular race track the robot will finish its exploration round at the starting position and can therefore utilize Loop Closure Detection to effectively improve the accuracy of the created map.

B. *Track Detection*

While the SLAM algorithm is useful for mapping the unknown environment and using this data for localizing the robot inside of it, the obtained map only categorizes a certain position as either free, occupied or still unknown. To identify the track boundaries based on this data an object detection algorithm has been developed to identify the traffic cones. Since the track boundaries do not entirely consist of these traffic cones but are instead defined by the continuous line through said cones, a track detection technique has been implemented to reliably connect the previously detected cones to infer the track boundaries.

1) LiDAR-based Object Detection: The object detection algorithm works on the occupancy grid map created from the lidar data and is therefore subject to noise due to inaccuracies from the SLAM algorithm which itself is affected by the imprecise wheel odometry as well as noisy lidar data. For this reason, a robust and reliable object detection technique had to be developed in order to detect the traffic cones based on incomplete and inaccurate data. Additionally, due to timing constrains the object detection also had to fast in order to work well during exploration. The detected cone positions are then used to reconstruct the track boundaries which is described in detail in the following section II-B2. We noticed that the reconstruction of the track appears to be very difficult if even a small number of cones is not detected (false negative) while other objects which are wrongly detected as cones (false positive) can be filtered out later relatively easily. Therefore, we decided to configure the object detection to be rather aggressive and accept false positives in order to minimize false negatives. The implementation of the described algorithm is explained in detail in section III-A.

2) Track Boundary Reconstruction: The task of the this component is to infer the track boundaries and construct *invisible walls* between neighbouring cones which can then be used to guide the path and motion planning through the track. As mentioned in the previous section, this component has to use the spatial positioning of the detected cones to identify the cones which are important for the track boundary computation and ignore the ones which are not. The implementation of this algorithm is described in detail in section III-B.

3) Guiding the Track Exploration: While the main objective of the track boundary detection is to guide the robots motion planning it can also be used to effectively determine the outskirt of the currently detected part of the track during exploration. We use this information to compute the *exploration target*, a point near the outermost recognized traffic cones still belonging to the track which should yield new information if approached while exploring. This point can be used by the racing control unit (see II-F) as a goal for the global planner to effectively map the track during exploration mode. The exact computation of this point is described in III-C.

C. Path Planning

The main aim of path planning in mobile robots is to find a safe and optimal path [10]. The path planning can be divided into two components, global path planning and local path planning. Global path planning focuses on providing an optimal path to an goal further away. However global planning is not enough as we have motion constraints as well as dynamic obstacles in the real world. Therefore, these problems are tackled by local path planning that receives up to date sensor information and provides a collision free path considering the robot's motion constraints [11]. We used *move_base* that provides a connection between global and local path planners to achieve it's navigation tasks. Moreover, it has different already existing implementations for global and local planners. This enables us to choose between different global as well as local planners in order to suit our requirements to complete the navigation tasks as per the constraints of the robot [12].

D. Global Path Planning

Global planning is an important component in navigation operation of mobile robots. The global planner calculates a path to reach the desired goal. The main task of the global planner is to find a kinematic feasible path that is collision free from the start position to the desired goal position. The global planner can by-pass the dynamic or differential constraints and this part is taken care by the local planner. The input to the global planner is the map that is constructed using the SLAM algorithm. The map provides the data for the obstacles in order to avoid them [13].

The selected global planner defines the complexity of the path that the local planner has to follow to reach the desired goal. We tried out two global planners in order to meet our requirements and then we compared them to select the one that provides us with the desired results. Our global planner choices are as below:

1) SBPL_Lattice_Planner: It is a plugin for *move_base*, the paths are produces by merging a a series of "motion primitives" considering shortest distance and kinematic constraints of the robot. Planning is computed in x, y and theta dimensions. The generated plan results in a smooth path that takes the robot's orientation into consideration. The robot's orientation is an important factor while planning paths for non-holonomic robots. An illustration of simulation that demonstrates the computed smooth trajectory for a goal is shown in Fig. 2.

SBPL_Lattice_Planner provides an option to choose between ARA* and AD* as the planning algorithms [14]. We used the ARA*, since the race track or the map cells were not changing with time and the local planner can take care of the dynamic obstacles [15], [16].

2) Global_Planner: It is a plugin for *move_base*, that provides a fast and interpolated global planner for navigation. It provides implementations for two well known path planning algorithms, A* [17] and Dijkstra's Algorithm [18]. We used Dijkstra's algorithm. An illustration of the computed path for a goal using Dijkstra's algorithm in the global planner is shown in Fig.3.

Since robot's orientation is an important factor while planning paths for non-holonomic robots we added orientations to the points on the computed path. We selected interpolate as the orientation mode in order to have linear blend of orientations from the start point to the goal [19].

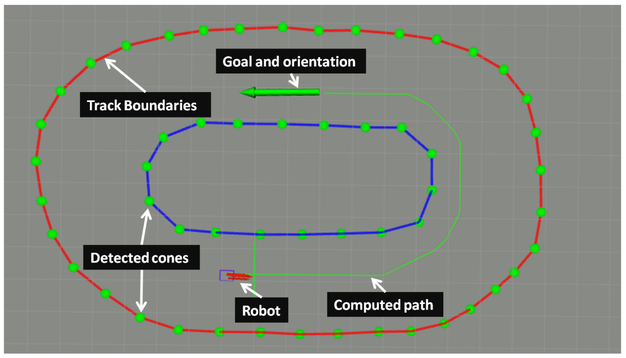

Fig. 2. Trajectories generated in simulations by the sbpl_lattice_planner

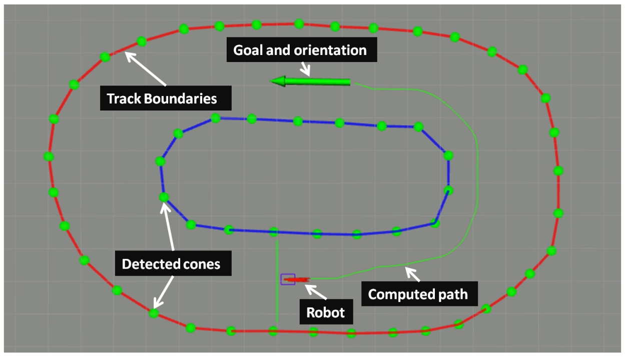

Fig. 3. Trajectories generated in simulations by using the Dijkstra's algorithm of global_planner

3) Algorithm choice: For our application of an autonomous race car we initially tried the *sbpl_lattice_planner* as it provided smooth paths with respect to the kinematic constraints of our robot. One of the challenges we faced was the computation time of the global path. On sharp curves such as a U turn, the planner consumed a lot of time to compute the path and this delay led to the robot crashing into traffic cones in simulations.

In comparison to the *sbpl_lattice_planner*, the *global_planner* provided a path faster than the *sbpl_lattice_planner*. The only concern that we suspected before running the simulations was that the *global_planner* might not provide a smooth trajectory as the *sbpl_lattice_planner*. But after running the simulations, we found that the generated trajectories by *global_planner* were smooth as shown in Fig. 3 and computationally faster. The local planner was able to follow the path generated by the *global_planner*. In combination with the local planner, the robot was able to take sharp turns such as a U turn more easily and without going off the track boundaries. It also provided better performance in terms of lap time over *sbpl_lattice_planner* as the computed paths were shorter and took less time to compute. Hence, due to fast computation and better performance we finally selected the *global_planner* for global path planning.

E. Local Path Planning

Once the global planner calculates the path that needs to be taken to reach the goal, it is then the job of the local planner to create a new path taking into consideration the different constraints of the vehicle and any dynamic obstacles that may appear on the path. The local planner needs to recalculate its path at a higher rate than the global planner since it needs to take into account these aforementioned dynamic obstacles. Hence, a local planner looks only at a small area of the map in the immediate vicinity of the vehicle.

The choice of local planner affects the complexity of the path computed for the vehicle to follow, the speed at which the vehicles moves and how well is it able to avoid dynamic obstacles. Our choices for local planner included the following:

1) Pose Follower: The simplest of all local planners is the pose follower which just attempts to follow the path computed by the global planner as closely as possible.

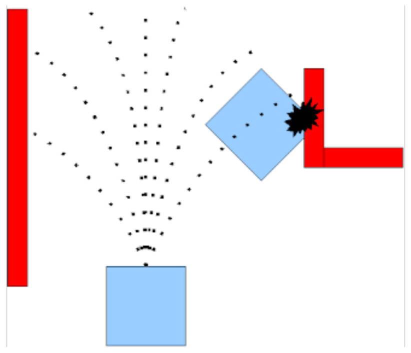

Fig. 4. Simulation of sampled trajectories by the dwa_local_planner [20]

2) Dynamic Window Approach (DWA): As the name suggests, the *dwa_local_planner* [20] uses a window of configuration space around the current position of the vehicle to create a grid of cells which contains the cost of travelling through the particular cells. These values are then used to sample different velocities and perform simulation on the vehicles for a short period of time to predict its behaviour as shown in Fig. 4. Each simulated trajectory is then weighed based on different characteristics like speed, distance from the goal and the obstacles and also how close is the trajectory to the global path. The trajectory with the highest score is then sent to the vehicle controller to be executed.

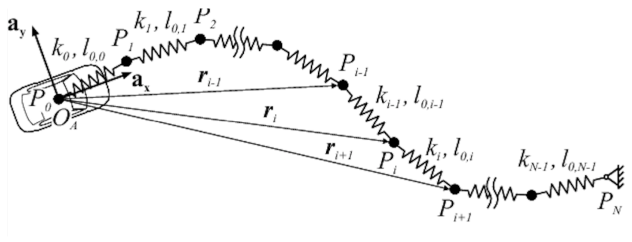

Fig. 5. Sample timed elastic band depicting the optimizations at intermediate points [21]

3) Timed Elastic Bands (TEB): The TEB [21] approach is designed with speed in mind. This planner optimizes the global path by calculating intermediate points to be taken by the vehicle while still keeping the velocity and acceleration limit constraints in mind. The paths generated by *teb_local_planner* are short and thus, theoretically, take the least amount of time to traverse. Fig. 5 shows an example of a timed elastic band for a vehicle and the different intermediate points at which the planner tries to optimize the path.

4) Algorithm choice: For our application of autonomous racing car, we initially decided to go ahead with the *teb_local_planner* since it was the one most optimized for speed. But, we faced a few issues with the *teb_local_planner* that could not be fixed.

The biggest challenge was our inability to disable backwards motion in the vehicle since it is impossible to do it due to its design. This resulted in our vehicle stopping and reversing for a short distance to go over a shorter path to the goal which would cost us time. This also resulted in time loss when the vehicle wanted to take a sharp turn.

On the other hand, the *dwa_local_planner* does not exhibit this problem. It has oscillation prevention built-in by design which makes it more tolerant to the path deviations. Also, by changing the value function used to weight the grid in the selected window, we are able to disable the reverse motion of the vehicle completely. This resulted in a local planner that prioritized speed, was more tolerant to deviations, and did not reverse during the race.

F. Racing Control

While the previously discussed components each solve one specific task our car has to solve we also needed one high level entity to connect and orchestrate these tasks and combine everything into a working race car. We called this component the *racing control* which has the main task of switching from exploration to racing mode after the initial mapping lap was completed and setting the goal for the global planner according to this mode. Furthermore, the motion commands generated by the local planner are modified by this component depending on the current mode the car is currently in (exploration or racing).

1) Exploration: The car starts the first lap in exploration mode. In this more the speed settings were chosen quite conservative because the time required for exploration was not important. Furthermore, the exploration target outputted by the track detection was directly fed into the global planner as goal as explained in II-B3.

2) Racing: The racing control continuously reads the published transformation tree to obtain the robots position and checks if the first lab was completed. After this is detected for the first time the current mode is changed to racing. In this mode the speed settings used to modify the motion control commands for the low-level speed and steering controllers are much higher and were carefully tuned to archive a fast lap completion time while simultaneously ensuring a safe driving behaviour without crashes.

We initially planned to compute a target point further in front of the robot in order for the global planner to be able to optimize the driving behaviour over a longer distance and thus resulting in smoother and more efficient movement. However, during testing we discovered that we were not able to optimize lab times or driving behaviour through this approach. Contrary to our expectations setting the goal farther away from the current position yielded worse trajectories since the global and local planner literally tried to cut corners and drove too close to the traffic cones marking the inner track boundaries. Since we were not able to tune the planners to archive our initial goal we stuck with the simple *exploration target* computed by the track detection and fed this into the global planner during the racing mode which resulted in feasible and reasonable fast trajectories.

3) Safety: As defined by the National Highway Traffic Safety Administration [4], a *Level 4 fully self-driving autonomous car* must be able to handle all safety-critical driving functions by itself. Since developing additional safety features was out of the scope of this project we decided to at least handle the safety measurements already in place in the used packages correctly. Currently the used local planner has two features which can be considered as safety features.

Firstly, the planner always tries to avoid obstacles during trajectory computation. Note that this is not limited to previously mapped obstacles but also includes any dynamic obstacle detected by the lidar. If no such trajectory could be found the planner sends a stop command (linear velocity equals zero) to the motor control. During modification of this command by the racing control stop messages are recognized and not altered. In addition to this feature if the local planner loses track of the robots position or exhibits any problems a stop command is also issued.

Since these stop messages are issued occasionally during normal operation we also implemented the option to overwrite them with regular speed commands. This notably increases the average speed both during exploration and racing, however the described features are deactivated and safety can not be guaranteed. This option should only be used with extreme caution and and in a controlled environment.

III. Implementation

A. Object Detection

This section aims to explain the implementation of the implemented traffic cone detection. The occupancy map obtained by the SLAM algorithm is taken as the only input for this task. The map consists of a two dimensional grid where each cell categorizes a certain position as either free, occupied or still unknown. The technique used to detect the cones is based on an image kernel approach where a matrix is applied to every pixel of an image and its neighbours similar to a convolution operation.

However, in our case we did not use a simple matrix but instead chose to determine the likelihood of a cone being at the position in question by a simple set of rules. These rules are based on the three categories a cell can be: free, occupied or unknown. Furthermore, neighbouring cells of the current pixel are split into two distinct areas: the center region directly around the pixel and the border, a rectangular stripe farther away from the center. These areas are visualized in Fig. 6 as red and blue respectively while free cells are marked as white, occupied cells as black and unknown cells as grey.

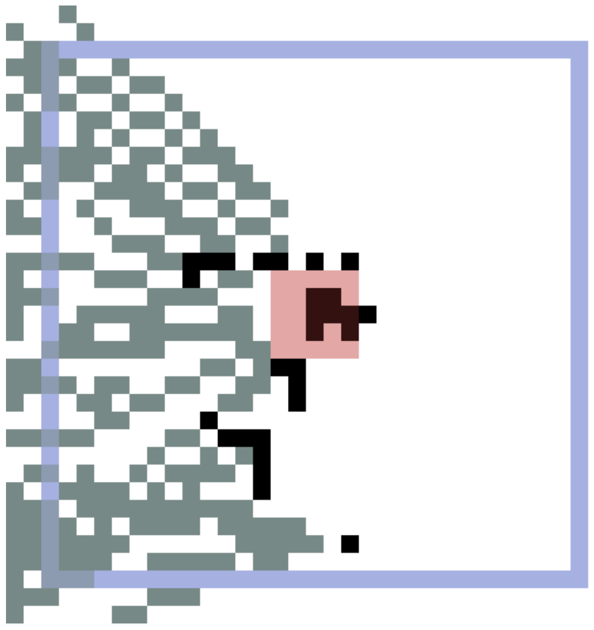

Fig. 6. Visualized image kernel over a detected cone

We then defined the following rules which must be true for the pixel to be identified as a possible cone:

- Number of occupied cells in the center must be larger than $minOccupiedCenter$

- Number of occupied cells in the border must be smaller than $maxOccupiedBorder$
- Number of unknown cells in the border must be smaller than $maxUnknownBorder$

These rules allow us to identify likely positions of isolated objects smaller than a predefined size and are applied to every cell. The dimensions of those areas as well as the thresholds are easily configurable. If the kernel matches the position is saved as a likely cone position. However, to actually be detected as a cone at least three kernel matches must lie closely together.

B. Track Boundary Reconstruction

The track detection algorithm is based on the assumption that adjacent cones belonging to one side of the track are not farther away from each other than the predefined $maxConeDistance$ parameter. Furthermore, the distance between cones belonging to opposite sides of the track consequently must be greater than this $maxConeDistance$ (which represents the track width in this case).

After receiving the cone positions from the object detection algorithm a greedy approach is used where every cone is compared with every other cone and all cones which are closer than the $maxConeDistance$ are added to the same group. Since we also assume that the robot starts inside of the racing track we now search for the two groups which contain the two cones closest to the left/right of the starting position, these groups are saved as left/right track boundary. Afterwards, invisible walls are generated between all cone pairs for each group and these walls are then added to the occupancy grid map as obstacles using a modified version [22] of Bresenham's Line Algorithm [23] for rasterization.

C. Exploration Target Computation

Using the two cone groups for the left and right track side computed in the previous section we can now easily come up with an exploration target position. We extend the robots position forward by a bit and starting from this position we again search for the two closest cones, one from the left track border and one from the right. To obtain the exploration target we now connect those cones and starting from the center point again extend forwards orthogonally to the connection line. This process is visualized in Fig. 7

D. Racing Control

As explained in II-F, the racing control modifies the motion commands generated by the local planner. The motion commands used consist of two values: angular and linear speed. The angular speed is not altered significantly and is only constrained to not exceed a defined range.

The linear speed value is then computed based on the steering angle without using the value generated by the local planner. For interpolating the speed the formula

$$linear\ \ speed = e^{-abs(angular\ \ speed)} \cdot max\ \ speed$$

is used which is also visualized in Fig 8.

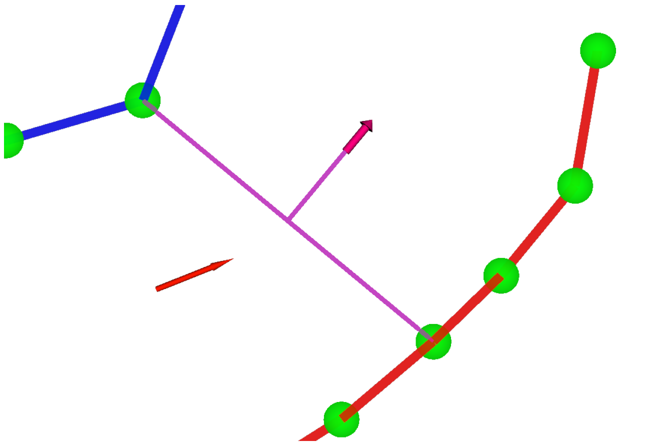

Fig. 7. Computing the exploration target (top right arrow) based on the robots pose (bottom left) and the center point (pink line) of nearby cones (green circles).

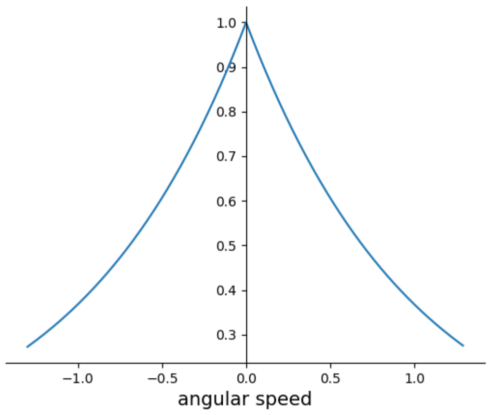

Fig. 8. Calculation of linear speed based on angular speed for $max\ \ speed = 1$

The linear speed is then constrained to the interval $[min\ \ speed,\ max\ \ speed]$ except when the local planner issues a stop command (linear speed equals zero) in which case the linear speed is set to zero (as long as the safety option is not disabled).

IV. Evaluation

A. Simulation Environment

Testing the small-scale vehicle in real-world requires time and manpower. This is a limitation to the number of experiments that can be done. To solve this, our solution was firstly tested in simulation where we tried to identify problems in order to ensure that they will not happen on the real car and thereby limiting the number unexpected issues during testing of the real car. The used simulator is MORSE - a generic simulator for academic robotic scenarios. However, during the transition from the simulation to real world we encountered many problems including different behaviour for previously tested speed parameters and noise affecting the lidar data in ways different to the noise we already anticipated and simulated in MORSE which required re-tuning of said parameters.

B. Results

Lastly, we evaluate the performance achieved during ten lap runs in our simulation environment on the already mapped track during racing mode. The track is outlined in figure 9. The starting line is indicated by the green line and the lap is finished when the car crosses this line.

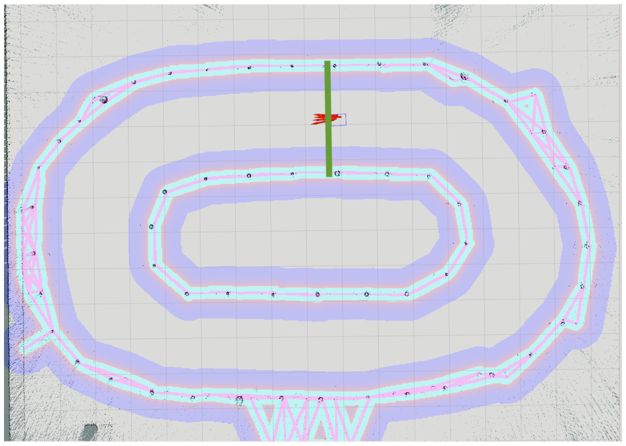

Fig. 9. Track used in the simulation.

The vehicle autonomously drove ten laps and we recorded times and noted down when the vehicle drove out of track as failure. In Fig.10 we indicated the average lap time for different speed parameters using the previously mentioned $dwa\_local\_planner$ in combination with $global\_planner$ and compared it to the $teb\_local\_planner$ in the combination with $global\_planner$. As result we can see that setting a higher speed results in a decreased lap time but taking this too far results in failures of the path planners and the car drives out of the given track. Also it's clear that on average the $dwa\_local\_planner$ performs faster than the $teb\_local\_planner$.

We also compared different combinations of $dwa\_local\_planner$ and $teb\_local\_planner$ respectively with $sbpl\_lattice\_planner$ for global path planning as shown in figure 11. As before, we can clearly see that using a higher velocity results in faster rounds but way more instability of the $global\_planner$, thus resulting in a failure to complete the lap by the robot. Compared to the $global\_planner$, the $sbpl\_lattice\_planner$ tends to fail more often in our track and therefore is the less optimal solution. As both of the path followers using $sbpl\_lattice\_planner$ have high failure rate it is difficult to make any conclusion from the estimated lap times.

C. Achievements

Our final goal was to implement a self-driving race car which could drive on a previously unknown track mapped only by using a lidar sensor. The most challenging part was to correctly estimate the track without using any input from camera, for which the previously mentioned Track Boundary Reconstruction algorithm was developed. We encountered unexpected issues during the selection and integration of the path planners (mentioned in IV-A), but these were at the end successfully solved as described in the previous sections.

Over the course of the past five months, the development progress of the self-driving small-scale vehicle was significant and most of our set goals were achieved as described in the previous sections and showcased during the final presentation.

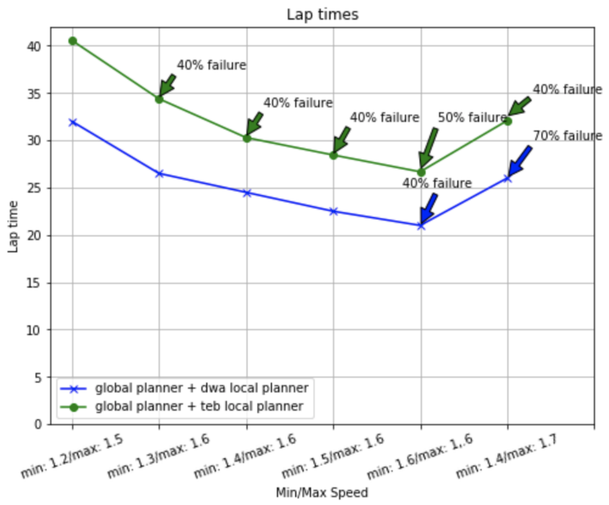

Fig. 10. Comparison of average times per round (y-axis) in the simulation environment with given velocity parameters (x-axis) using combination of dwa_local_planner and global_planner vs. teb_local_planner and global_planner

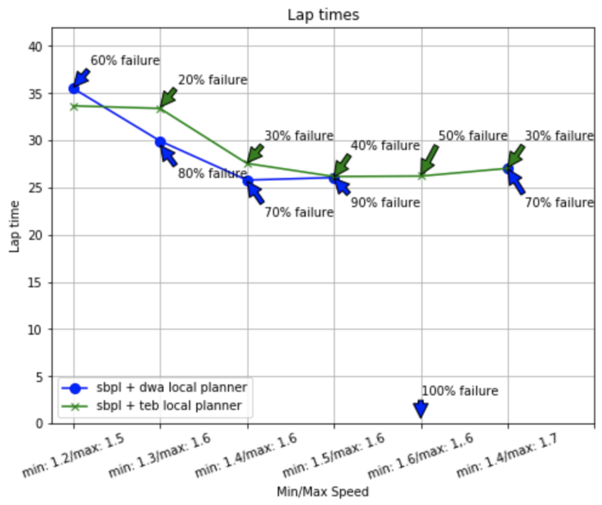

Fig. 11. Comparison of average times per round (y-axis) in the simulation environment with given velocity parameters (x-axis) using dwa_local_planner and sbpl_lattice_planner for global path planning vs. teb_local_planner and sbpl_lattice_planner for global path planning

V. Conclusion

A. Future Work

Given the restricted time frame, we were able to achieve working solution, but there is still some things we want to improve.Firstly, we want to focus on improving the robustness, meaning that the car should be able to race on more complicated tracks, which it haven't seen before without drastically modifying parameters or already implemented algorithms. We are also planning to test the car using higher velocity, especially on the real car, if we will be able to. Also we were trying to come up with alternative solution to traditional robotic techniques towards the end of the project, but given the problems with transitioning from simulation to real environment and spending last months selecting the optimal path planer algorithms, this effort toward the more machine learning solution was mostly neglected. The idea was to optimize the velocity and car position using advanced deep reinforcement learning techniques presented in [24]. Although we were able to implemented working base for this algorithm based on wall following, it is still not stable enough to start experimenting with parameter optimization. Therefore in the coming months we plan to focus on improving stability of our global planners or implementing stable solution of wall following which could be further optimize.

B. Lessons Learned

We would like to share some of lesson gained from the experience of working in the team of students developing on self-driving car.

Firstly, we have discovered how important hardware is, and only through testing in real environment one can roughly evaluate such a complex system as a whole. Therefore, testing is the most vital factor for success. Simulation testing is also invaluable, as it allows to make real-world testing more efficient and prevents us to make unnecessary mistakes.

Having a clear software design with defined interfaces makes it possible to develop and work, and test each subsystem independently and therefore distribute the task more efficiently. This also helps with having a well structured group and responsibilities of every team member. Using a good version control system such a gitlab is necessary for good collaboration and also for project management. Thus making the communication between team members easier.

Finally, we have learned to work with on the real world small-scale car outside of the simulation, which was invaluable experience as it motivated us to have a working solution which could be presented during the final weeks and kept us engaged throughout the whole semester. This is an opportunity which is rare in the studies and gave us vision of our future opportunities.

References

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[2] Carnegie Mellon University, "Navlab: The Carnegie Mellon University Navigation Laboratory," https://www.cs.cmu.edu/afs/cs/project/alv/www/index.html.

[3] L. Fridman, "Sebastian thrun: Flying cars, autonomous vehicles, and education — artificial intelligence podcast," https://www.youtube.com/watch?v=ZPPAOakITeQ, accessed: 2020-02-01.

[4] National Highway Traffic Safety Administration, "Preliminary statement of policy concerning automated vehicles," https://cutt.ly/erFYCb3, accessed: 2020-02-02.

[5] C. Hrabia, "Lecture notes in applied artificial intelligence project," October 2019.

[6] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[7] J. M. Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2d slam techniques available in robot operating system," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR).* IEEE, 2013, pp. 1–6.

[8] B. Gerkey, "slam-karto," https://github.com/ros-perception/slam_karto.

[9] SRI International, "Open karto," https://github.com/skasperski/OpenKarto.

[10] R. C. G. L. Hassani Imen, Maalej Imen, "Robot path planning with avoiding obstacles in known environment using free segments and turning points algorithm," p. 13, 2018.

[11] Y. Lu, X. Zhucun, G.-S. Xia, and L. Zhang, "Design and performance analysis of global path planning techniques for autonomous mobile robots in grid environments," *International Journal of Advanced Robotic Systems*, vol. 14, no. 2, 2018.

[12] Open Source Robotics Foundation, "move_base," http://wiki.ros.org/move_base.

[13] I. Chaari, A. Koubaa, H. Bennaceur, A. Ammar, M. Alajlan, and H. Youssef, "A survey on vision-based uav navigation," *Geo-spatial Information Science*, pp. 1–12, 01 2017.

[14] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm." AAAI Press, 2005, p. 262–271.

[15] Open Source Robotics Foundation, "sbpl_lattice_planner," http://wiki.ros.org/sbpl_lattice_planner.

[16] N. Limpert, S. Schiffer, and A. Ferrein, "A local planner for ackermann-driven vehicles in ros sbpl," 2015.

[17] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.

[18] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec 1959. [Online]. Available: https://doi.org/10.1007%2Fbf01386390

[19] Open Source Robotics Foundation, "global_planner," http://wiki.ros.org/global_planner.

[20] O. S. R. Foundation, "dwa_local_planner," http://wiki.ros.org/dwa_local_planner.

[21] T. Sattel and T. Brandt, "Ground vehicle guidance along collision-free trajectories using elastic bands," *Proceedings of the 2005, American Control Conference, 2005.*, pp. 4991–4996 vol. 7, 2005.

[22] A. Zingl, "The beauty of bresenham's algorithm," http://members.chello.at/~easyfilter/bresenham.html.

[23] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems journal*, vol. 4, no. 1, pp. 25–30, 1965.

[24] Autonomous Racing Project Group of TU Dortmund, "Autonomous racing software stack and simulation enviroment," https://github.com/Autonomous-Racing-PG/ar-tu-do.