TECHNISCHE UNIVERSITÄT WIEN

Autonomous Racing Cars
191.119 (VU 4,0) Semester: 2022S

Mapping and Localization

2022-04-19

Student data

Matrikelnummer	Firstname	Lastname
12134689	Danilo	Castiglia
12135191	Artur	Chaves
01613004	Severin	Jäger
01552500	Johannes	Windischbauer

Transparency of contribution

- **Danilo Castiglia**: ROS bag recording, offline SLAM mapping both with simulator and hardware car, ROS bag splitting.
- **Artur Chaves**: Cartographer
- **Severin Jäger**: Sensor fusion, Cartographer setup, Simulator modifications for AMCL
- **Johannes Windischbauer**: gmapping attempts, ROS bag recording, AMCL

Note

This page will not be shared with other teams. All following pages and the submitted source code archive will be shared in TUWEL after the submission deadline. Do not include personal data on subsequent pages.

TECHNISCHE UNIVERSITÄT WIEN

Autonomous Racing Cars
191.119 (VU 4,0) Semester: 2022S

Mapping and Localization

2022-04-19

Team data

Team number: 2

1 Creating a Map

As a team member had prior experience with the gmapping package we decided to go with it. The first step was to install required packages:

```
sudo apt-get update
sudo apt-get install ros-noetic-gmapping ros-noetic-map-server
```

1.1 SLAM

Firstly, we tried to use the gmapping package to record the maps, but the results were not satisfactory even after tuning them, so we changed to the Google cartographer[1] to see if the results were more satisfying. We used the installation tutorial from the slides to proceed to the install of the cartographer. A new work space was created to install the cartographer, called `cartographer\_ws`. After the building the cartographer it was needed to configure the robot to the model of the car. To do this we used files from the official f1tenth lab repository[2]. After that everything was set to start mapping.

a.) Google cartographer builds up local submaps and merges them together. The submaps are extremely accurate (at least with the odometry in the simulator), but merging is a hard process and can only succeed with loop closure. In general, this is (relatively independent of map and our parameter modifications) extremely computation-hungry. Real-time visualisation is only feasible on powerful destop systems.

The first time driving the car with the cartographer to record the track it was used the default configuration from the car model. The car was driving with the wall follow node with the following parameters: $v_{max} = 1.5\,\mathrm{m/s}$, $v_{min} = 0.5\,\mathrm{m/s}$, $\theta_{lidar} = 35°$.

The default configuration of the cartographer worked decently to map the track f1_aut_wide, however we modified some parameters slightly:

Parameters	Default	Optimisation
`TRAJECTORY_BUILDER_2D.max_range`	25	30
`TRAJECTORY_BUILDER_2D.min_range`	0.5	0.06
`TRAJECTORY_BUILDER_2D.submaps.num_range_data`	100	150
`POSE_GRAPH.optimize_every_n_nodes`	20	5

Table 1: Optimization of the parameters

Mainly, we adjusted the laser scanner ranges. Even though this is limited to simulation, increasing the laser scanner range helped a little with very long straight lines (e.g. in the f1_esp track). Furthermore, we increased

[1] https://google-cartographer-ros.readthedocs.io/en/latest/index.html
[2] https://github.com/f1tenth/f1tenth_labs/tree/main/cartographer_config

the size of submaps which reduces the number of submaps. This simplified the loop closure in long maps. As we had significant troubles with the f1_esp track, we increased the number of cartographer optimisation steps as well.

b.) The previous task as completed in three tracks: f1_aut_wide, f1_mco and f1_esp. At the f1_aut_wide as shown at the Figure: 1 it can be seen that the cartographer worked well, the tracks look similar when running 1 or 3 laps. At the f1_mco, a track that has more complex turns and the cartographer was still running well, all 3 tracks look similar figure (2), but when the car runs more laps the track quality improves. In the first image 2a there is a part that looks like a straight and then with the 3 laps figure (2c) we can see more clearly a turn there. The last map f1_esp is the one that had more problems when trying to build the map, has seen int the figure 3 the track is overlaying in the first turns and in the straight line the cartographer had problems, the straight is to big and it started to turn to soon and sometimes did not close the loop. For this track the tuning has to be improved to get better results like in the previous two tracks. When changing the starting position the loop will close at a different position, due to this a better map build could be achieved if the loop closer is easier in the new position, so the start position can influence the quality of the map.

Google cartographer can consider laser scan data, odometry, and IMU data to calculate SLAM[3]. As the f1tenth simulator does not provide reasonable IMU messages, we disabled the IMU channel in the cartographer configuration script[4].

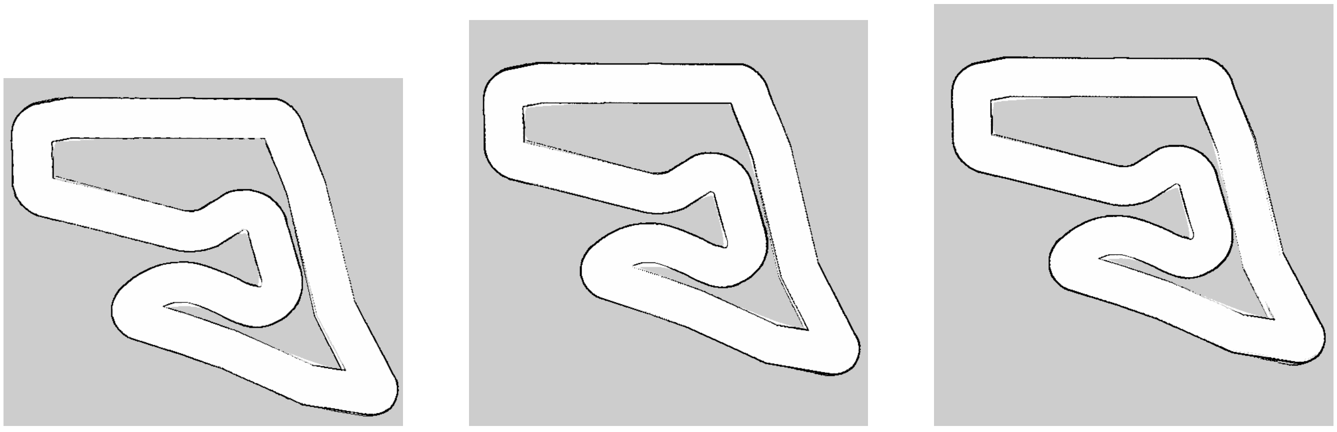

(a) 1 Lap (b) 2 Laps (c) 3 Laps

Figure 1: Differences between 1, 2 or 3 laps around f1_aut_wide

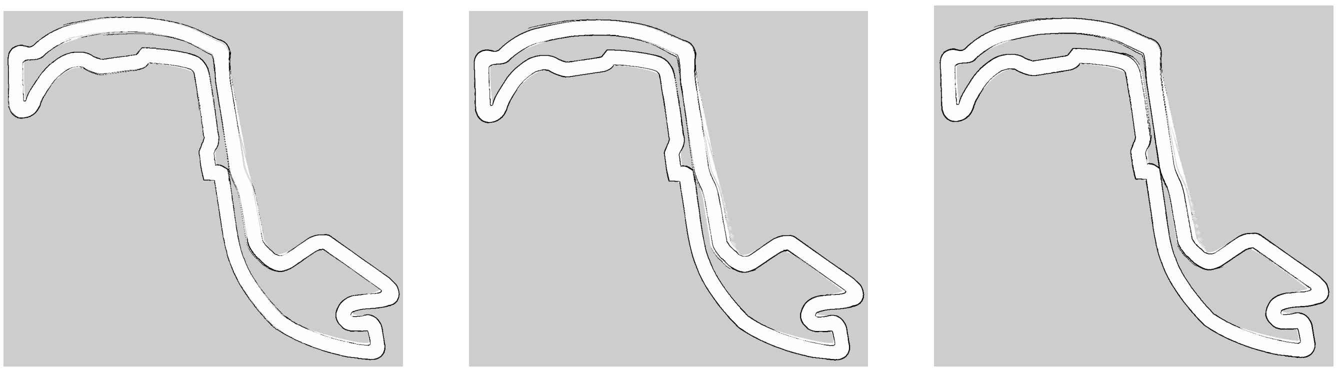

(a) 1 Lap (b) 2 Laps (c) 3 Laps

Figure 2: Differences between 1, 2 or 3 laps around f1_mco

[3]Block diagram at https://google-cartographer-ros.readthedocs.io/en/latest/algo_walkthrough.html
[4]f110_2d.lua

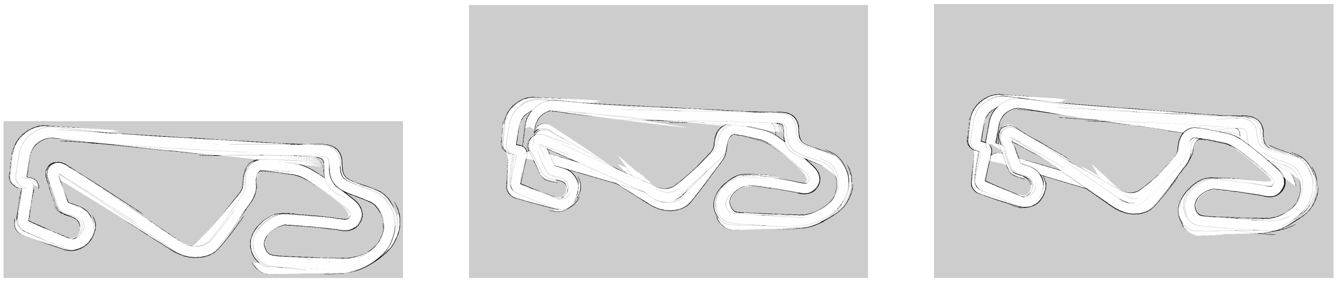

(a) 1 Lap (b) 2 Laps (c) 3 Laps

Figure 3: Differences between 1, 2 or 3 laps around f1_esp

c.) Using the map_saver the maps of the task b) were saved and then compared with the original ones Figures:4, 5 and 6. As seen in the Figure:4 the recorded map is very similar to the original map, with very little differences. With this configuration this map could be recorded and then used for navigation.

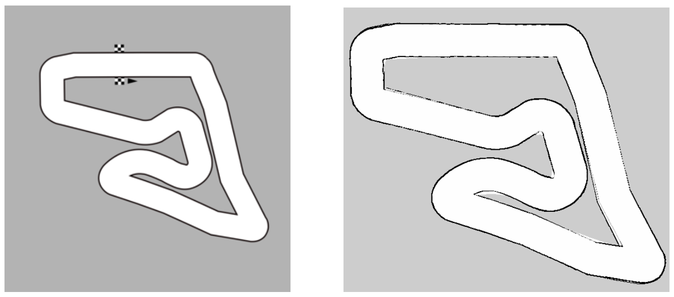

(a) Original Map (b) Recorded Map

Figure 4: Comparison between the original map and the recorded map at f1_aut_wide

In the map f1_mco seen in the figure:5 which has more sharp turns and some turn sequences, it was still possible to create a map that looks like the original, but in this tracks is possible to see a portion where the track seems to have some deviation comparing with the original. When looking closely, where is suppose to be turn 2, comparing both images, the figure:5b does not have the turn shaped in the same way has the figure:5a. With more tuning of the parameters referred in the topic 1.1 a) the track could get better in that portion.

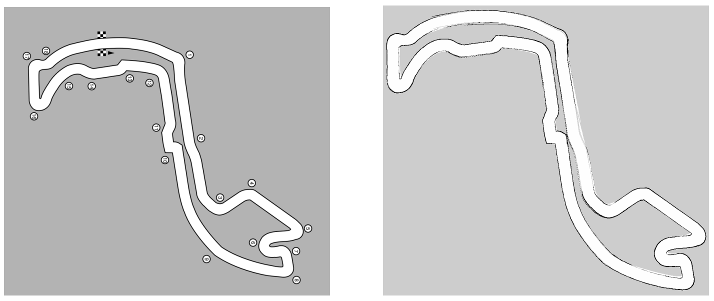
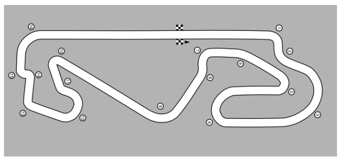

(a) Original Map (b) Recorded Map

Figure 5: Comparison between the original map and the recorded map at f1_mco

In the map f1_esp seen in the figure:6 there are some problems with the recorded map 6b mainly in the straight line and the first turns of the map. In the straight line has some difficulties closing the loop that is why it does not look straight, the problem seems to be the length of the straight line which lacks distinctive features. This recorded map would not be good enough to use in the navigation algorithm due to this differences.


(a) Original Map

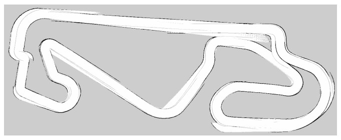

(b) Recorded Map

Figure 6: Comparison between the original map and the recorded map at f1_esp

d.) The workflow to record a rosbag from the simulator and to run it in cartographer is:

1. Launch the simulator: `roslaunch wall_follow wall_follow.launch`.
2. In another terminal start the recording of the rosbag: `rosbag record /scan /odom /imu`.
3. To stop the recording and save the bag file we can just use `CTRL + C`.
   The output file is `2022-05-03-14-38-05.bag`.
4. Now we have our rosbag and we want to create our map offline. We start Cartographer from its workspace:
   `roslaunch cartographer_ros f110_2d.launch`
5. In a second terminal we play the rosbag: `rosbag play 2022-05-03-14-38-05.bag`
6. When the rosbag play is terminated, in a third terminal we save the map with:
   `rosrun map_server map_saver -f offline_simulator_map map:=/map_slam`

We obtain the map as 2 output files, a PGM: the image of the map, shown in Figure 7 and a YAML file: the description of the map. Before launching cartographer_ros it is needed to run the source command:
`source ~/cartographer_ws/install_isolated/setup.bash`.

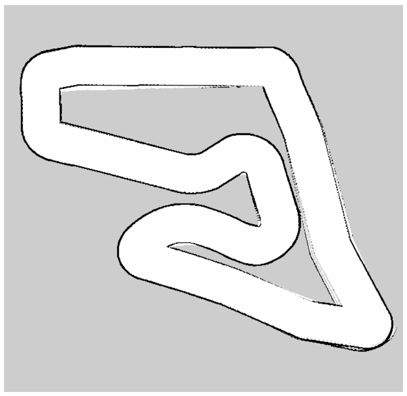


Figure 7: Map of the f1_aut_wide track obtained using SLAM offline with 3 laps.

e.) One way to split the rosbag in 2 or more sections is to use the rosbag filter commands twice. For example, to divide the rosbag in 2 equal parts, based on the time, we can run this command:
`rosbag filter input.bag output1.bag "t.to_sec() <= 1284703931.86"` followed by:
`rosbag filter input.bag output2.bag "t.to_sec() > 1284703931.86"`.
For convenience purposes it can be automated with a simple bash script like the following:

```
#!/bin/bash
# provide input bag, output bags prefix, and time fraction
```

```
echo $1, $2, $3
t0=`rosbag info -y -k start $1`
t1=`rosbag info -y -k end $1`
tfr=`echo "$t0 + ($t1 - $t0) * $3" | bc -l`
echo $t0, $t1, $tfr
rosbag filter $1 $2_a.bag "t.secs <= $tfr"
rosbag filter $1 $2_b.bag "t.secs > $tfr"
```

Splitting the bag in 2 parts doesn't make any difference in the produced map, it is almost identically to the map in Figure 7. This is because we made 3 laps and we split in 2 bags, so we have 1.5 laps in each bags. Instead splitting the 3 laps of the bag in 4 parts produce 4 partial maps. We can see one fo them in Figure 8.

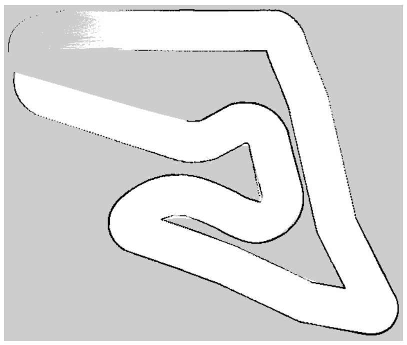


Figure 8: Map of the f1_aut_wide track obtained using SLAM offline with 1 section out of 4 and 3 laps.

f.) **TODO:** HW

g.) **TODO:** HW

2 Localization

2.1 Particle Filter

Visualization of a particle filter for a stretch around track.

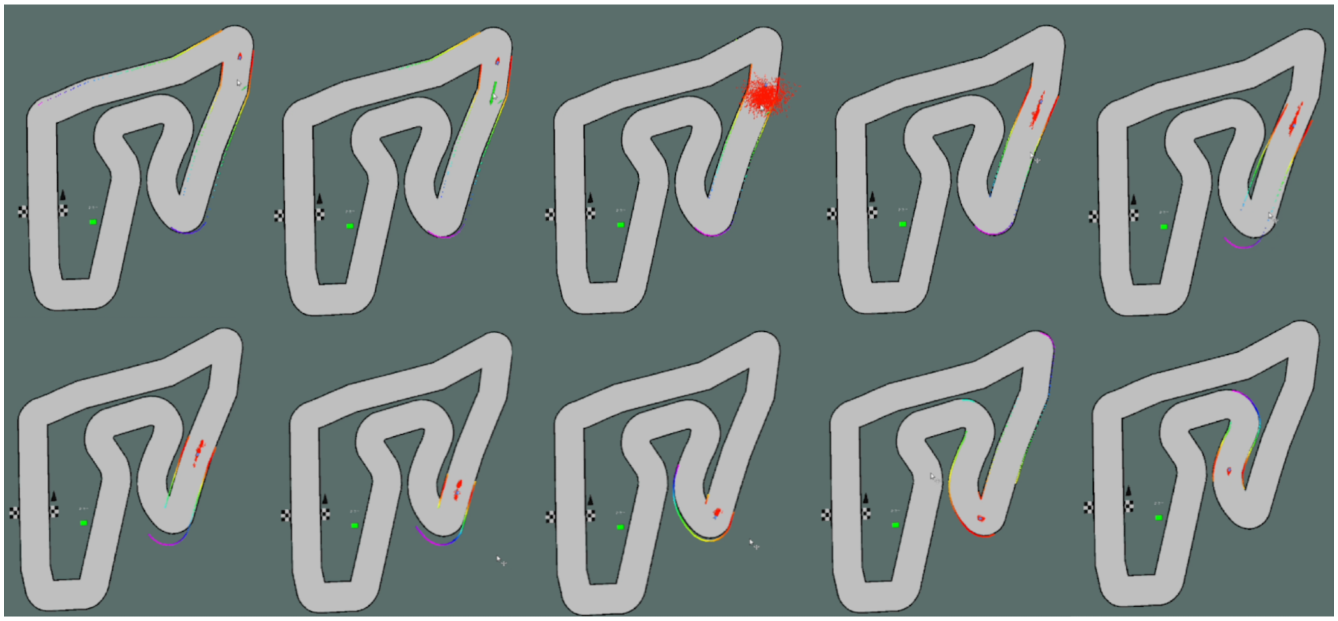


Figure 9: From left to right in each row: Advancements over time when resetting the particle filter position in frame two. Frame 3 shows particles distributed around the newly set point. From frame 4 on-wards the particles shrink back to merely a point. The red points are the particles; the little green arrow (frame 2) is the manual setting for a 2D pose estimate.

a.) We used the AMCL [5] particle filter that is referenced in the assignment. To install it in our environment it was necessary to execute the following commands:

```
sudo apt-get update
sudo apt-get install ros-noetic-amcl
```

After installing the AMCL node can be executed by `rosrun amcl amcl`. However, our standard simulator and AMCL config did not work. We figured out that we can change the required frames from the AMCL node which resulted in us getting the first visible particles, but with a wrong TF tree. Subsequently, we tried introducing another frame in the simulator, but this did not seem very practical regarding the other assignments. Therefore, we reverted those changes and resorted to our final resolve, which was making the simulator map_frame_id configurable via a parameter. How the particle filter looks in RViz, once the PoseArray visualization is added and subscribed to the particle cloud can be seen in figure 9 and in figure 10. In regard to tuning particle filter parameters, there are quite some modifications but probably the most noticeable are slight changes to update_min_a and update_min_d and increasing the number of particles by the factor of two. Tuning the particle filter is a difficult topic and the standard configuration worked quite well already. Overfitting the particle filter is not desirable since a the filter cannot determine the pose correctly without enough randomization once the right position is lost due to various reasons. The full AMCL configuration can be found in listing 1.

[5] http://wiki.ros.org/amcl

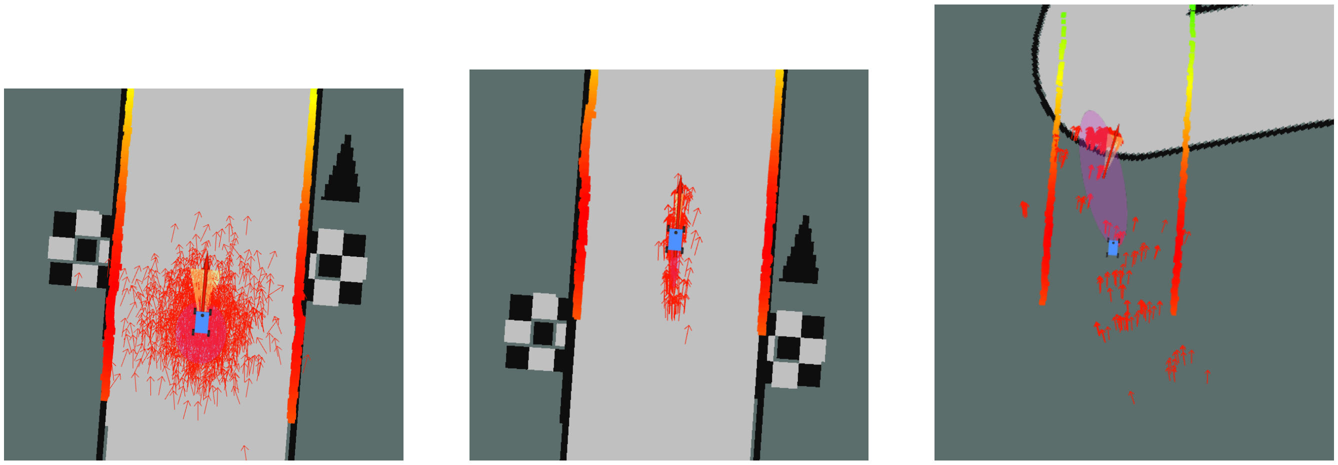

(a) Start up (b) Early driving (c) Bad (manual) pose estimation

Figure 10: Pose array visualizations in different stages of driving

b.) The AMCL particle filter utilizes the laser sensor data and the known map for localization. Furthermore, it subscribes to the tf topic and accepts an initial pose for start up and relocation. The node then publishes a transformation between map and odom. The point cloud is can be visualized in RViz using the PoseArray, as previously stated and can be seen in figures 10 and 9. For this task we chose the f1_aut_wide, the f1_aut_wide_slam, f1_mco and f1_mco_slam maps. The self-localization has a harder time to locate the position if the recorded maps are not really accurate and contain a lot of similar stretches. Both maps as ground truth and as slam variant resulted in a pretty decent localization as can be seen side by side in the following picture series.

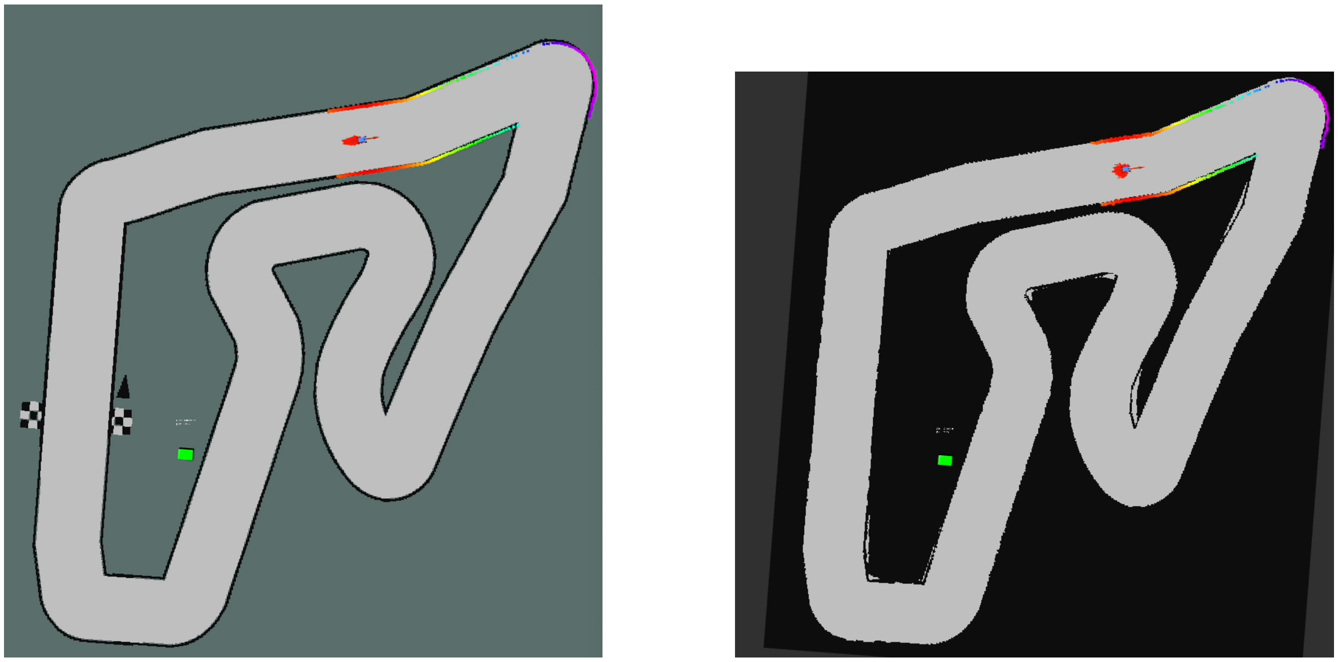
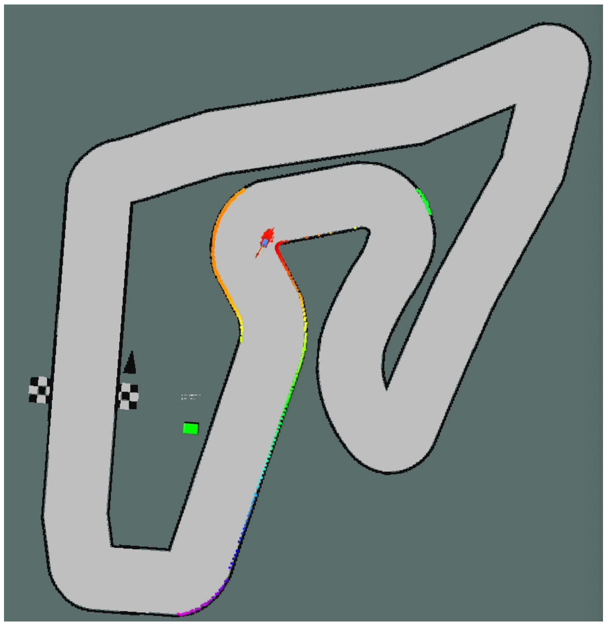

(a) ground truth (b) slam

Figure 11: AMCL on the f1_aut_wide track on the longest straight


(a) ground truth

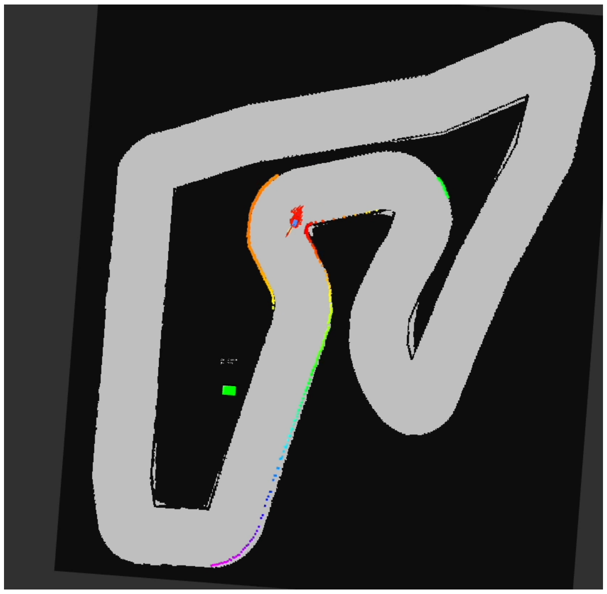
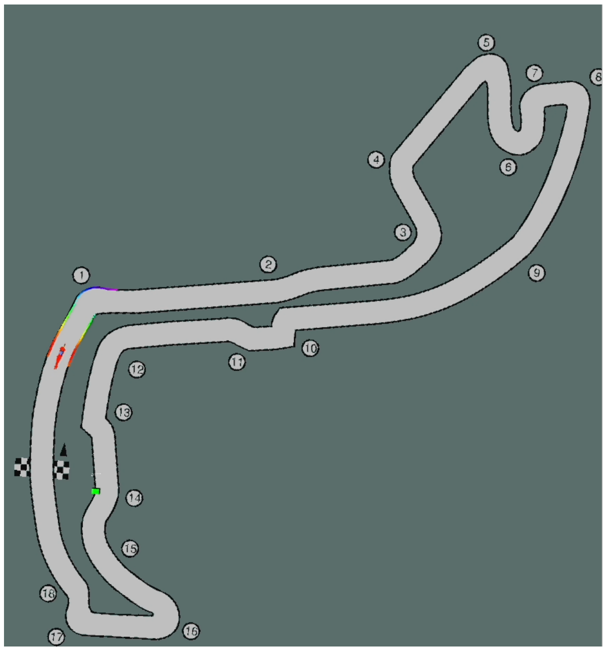

(b) slam

Figure 12: AMCL on the f1_aut_wide track in a sharp corner



(a) ground truth

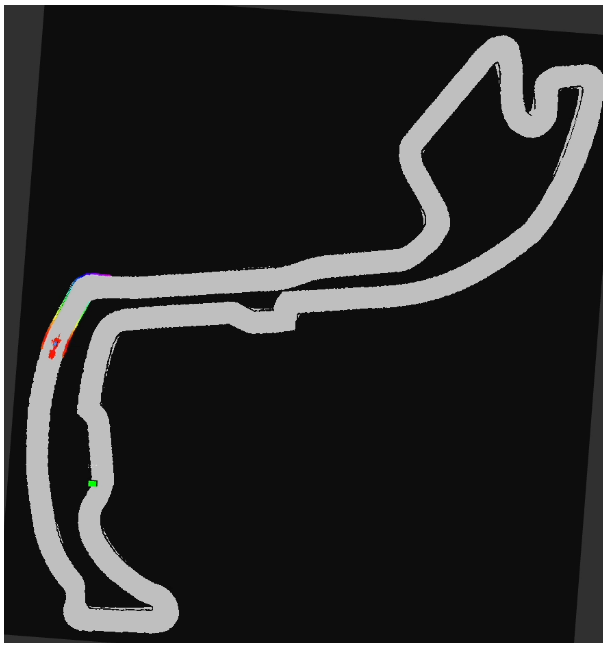

(b) slam

Figure 13: AMCL on the f1_mco track

c.) As already stated in the previous section the localization works quite well in general. However there are some situations where it is less than ideal or even stops working completely. These situations come up in a variety of situations. Minor errors usually result from small differences in the mappings and the sensor data from the car. Major errors usually resulted from manually setting an incorrect pose estimate. Some of these situations can be seen in the following screenshots. As a general rule of thumb it could be said that a better map results in a better self-localization and discrepancies in the map result in discrepancies in localizing the position.

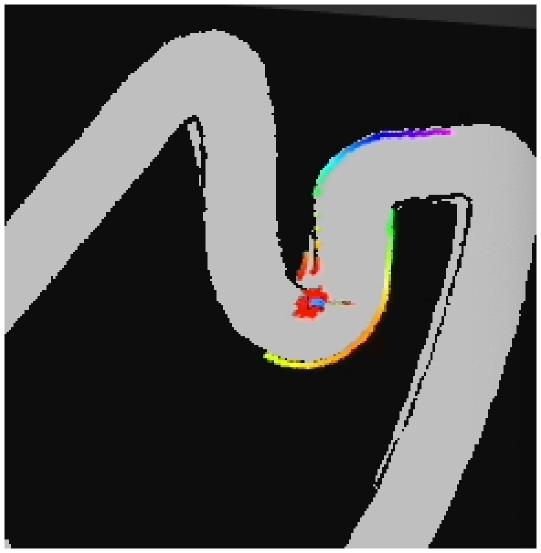


Figure 14: The robot has trouble following the left wall because of an incorrect mapping. This is more a mapping and autonomous driving problem than an actual AMCL problem.

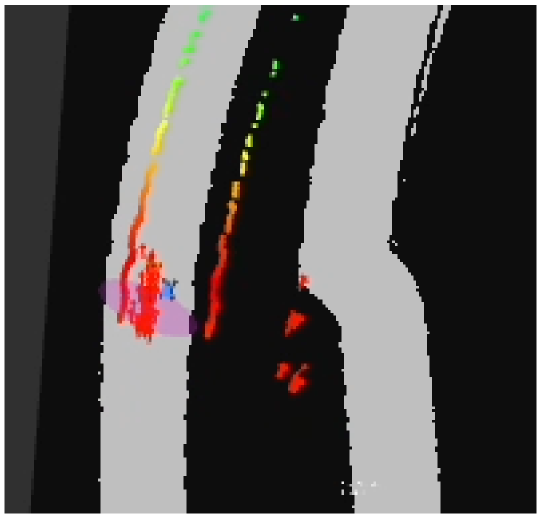


Figure 15: Localization and actual position of the car are not coinciding. The car is estimated to be a little bit further to the left.

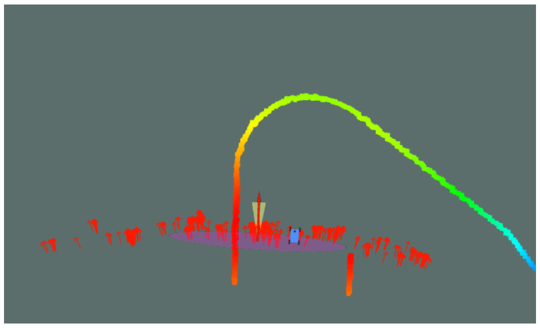


Figure 16: Due to setting the pose estimation manually to a wrong position and direction the particle filter was not able to recover and the car seemingly drove outside the barriers.

d.) **TODO:** HW

e.) **TODO:** HW

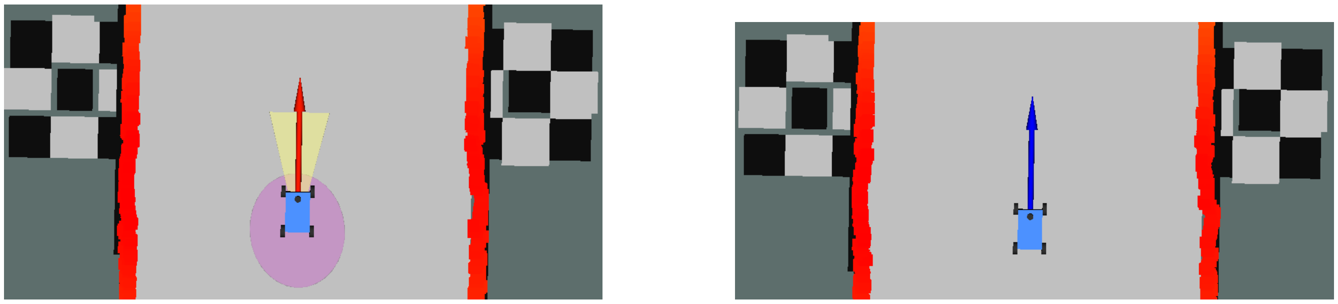
2.2 Improve the Localization

a.) As AMCL only considers the laser scans and the map, some sensor information is not considered. This includes the odometry (both in the simulator and on the car) as well as the IMU (unfortunately only on the real HW). The earlier can be used to determine the car's position with dead reckoning while integrated IMU information can also be used to estimate a position. Both channels are widely independent of the laser scan information and have their specific error patterns. Therefore, sensor fusion can be used to calculate the best position estimate out of all three measurements.

b.) We used an `ekf_localization_node` from the `robot_localization` package[6] to implement sensor fusion. This node subscribes to an arbitrary number of odometry, twist, IMU, and pose topics (each including covariances) and publishes the fused information to an odometry topic.

For the simulator, the EKF takes the pose calculated by AMCL and the simulator odometry as input. We consider the x and y positions as well as the yaw angle from both sources as we use the 2D mode.

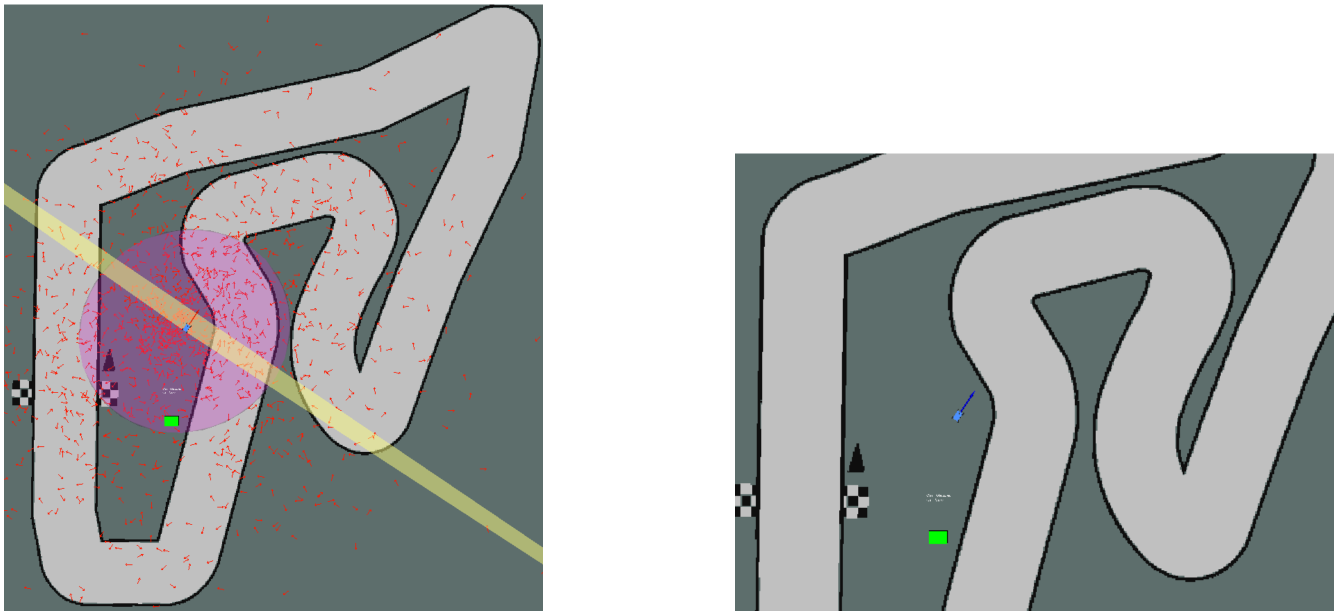
As the simutor odometry is effectively the exact position of the car, sensor fusion drastically reduces the position covariance, see Figure 17 and **??**. The fused position is directly the simulator odometry and thus the exact position. This somehow contradicts the idea of sensor fusion, this will however change on the car with imperfect odometry.



(a) AMCL Pose (b) Fused Pose

Figure 17: Sensor fusion at the start of the f1_aut_wide track

[6]http://docs.ros.org/en/noetic/api/robot_localization/html/state_estimation_nodes.html



(a) AMCL Pose (b) Fused Pose

Figure 18: Sensor fusion outside the f1_aut_wide track

c.) As the fused pose is now exactly the correct pose (without any variance), these tests can be used to assess the accuracy of AMCL (unfortunately not of the sensor fusion algorithm). One reproducible difference is the higher update rate of the odometry compared to AMCL. Thus, the AMCL pose is always lagging behind slightly, see Figure 19. In some parts of the SLAM maps, AMCL localisation is slightly off or show large variance, examples are given in Figures 20 and 21.

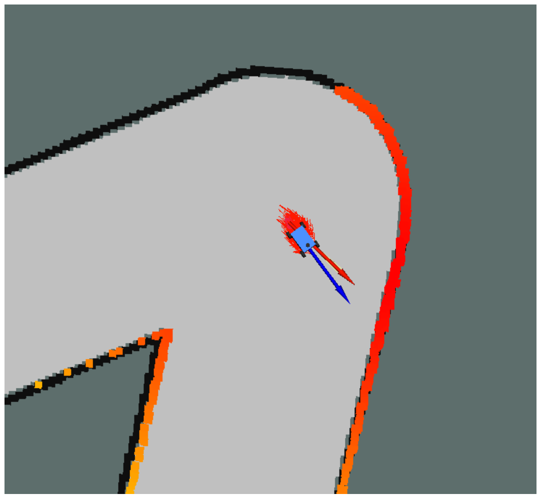


Figure 19: AMCL delay with AMCL pose (red) and fused pose (blue)

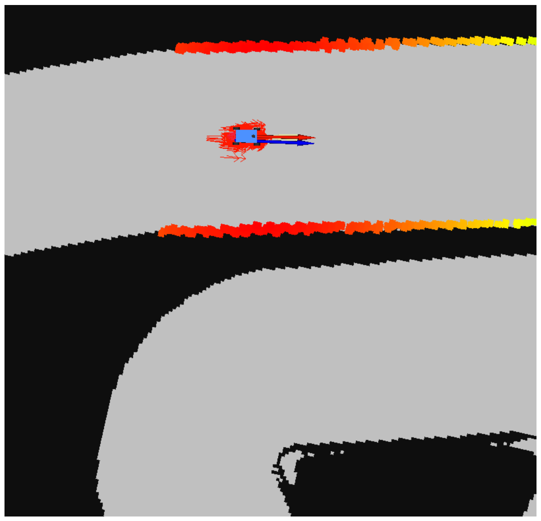


Figure 20: AMCL error in f1_aut_wide_slam map with AMCL pose (red) and fused pose (blue)

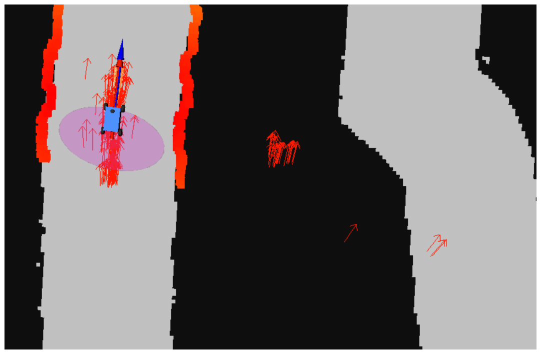


Figure 21: AMCL variance in f1_mco_slam map with AMCL pose (red) and fused pose (blue)

Unfortunately, the IMU simulation is not implemented. Thus, the more interesting fusion of IMU and AMCL can only be conducted on the hardware.

d.) **TODO:** HW

e.) **TODO:** HW

3 Appendix

Listing 1: AMCL launch file

```
<?xml version="1.0"?>
<node name="amcl" pkg="amcl" type="amcl" output="screen">
    <param name="use_map_topic"             value="$(arg␣use_map_topic)"/>
    <param name="odom_model_type"           value="diff"/>
    <param name="odom_alpha5"               value="0.1"/>
    <!-- Publish scans from best pose at a max of 4 Hz -->
    <param name="gui_publish_rate"          value="4.0"/>
    <param name="save_pose_rate␣"           value="4.0"/>
    <param name="laser_max_beams"           value="60"/>
    <param name="laser_max_range"           value="10.0"/>
    <param name="min_particles"             value="500"/>
    <param name="max_particles"             value="2000"/>
    <param name="kld_err"                   value="0.05"/>
    <param name="kld_z"                     value="0.99"/>
    <param name="odom_alpha1"               value="0.2"/>
    <param name="odom_alpha2"               value="0.2"/>
    <!-- translation std dev, m -->
    <param name="odom_alpha3"               value="0.2"/>
    <param name="odom_alpha4"               value="0.2"/>
    <param name="laser_z_hit"               value="0.5"/>
    <param name="laser_z_short"             value="0.05"/>
    <param name="laser_z_max"               value="0.05"/>
    <param name="laser_z_rand"              value="0.5"/>
    <param name="laser_sigma_hit"           value="0.2"/>
    <param name="laser_lambda_short"        value="0.1"/>
    <param name="laser_model_type"          value="likelihood_field"/>
    <!-- <param name="laser_model_type" value="beam"/> -->
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d"              value="0.25"/>
    <param name="update_min_a"              value="0.2"/>
    <param name="odom_frame_id"             value="odom"/>
    <param name="base_frame_id"             value="base_link"/>
    <param name="global_frame_id"           value="map"/>
    <param name="resample_interval"         value="1"/>
    <!-- Increase tolerance because the computer can get quite busy -->
    <param name="transform_tolerance"       value="1.0"/>
    <param name="recovery_alpha_slow"       value="0.0"/>
    <param name="recovery_alpha_fast"       value="0.0"/>
    <param name="initial_pose_x"            value="$(arg␣initial_pose_x)"/>
    <param name="initial_pose_y"            value="$(arg␣initial_pose_y)"/>
    <param name="initial_pose_a"            value="$(arg␣initial_pose_a)"/>
    <remap from="scan"                      to="$(arg␣scan_topic)"/>
</node>
```