TECHNISCHE UNIVERSITÄT WIEN

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

Lab 3: Follow the wall & Hardware Race Car

2022-03-29

Preface

Read all the instructions carefully before you start working on the assignment, and before you make a submission.

- Completeness of solution: A complete solution of a task also includes knowledge about the theory behind.
- Exercises are to be solved in teams. All team members must be indicated on the submission protocol. However, every team member must be able to explain the handed in solution. Grading is on an individual basis. Upload your solution (one per team) in TUWEL until 2022-04-19 23:59.
- For this assignment there is no exercise interview. However if submissions are unclear, students might be invited for exercise interviews.
- All sources of material and ressources must be properly cited.

Learning outcomes

The following fundamentals should be understood by the students upon completion of this lab:

- Work with PID controllers.
- Driving the car autonomously via wall following.
- Tuning algorithmns/controllers.
- Work with actual hardware in real-world-scenarios.

Deliverables and Submission

- Evaluate your implementation in the testbench in TUWEL (details see below).
- Write a exercise report and submit it as PDF file in TUWEL until 2022-04-19 23:59. Use the provided latex template and do not forget to fill in the parts marked with "TODO". If you do not use the provided template, make sure to include the same information. In any case the report must meet an adequate level for layout and readability that is appropriate for the academic context.
  If you are using any program or script (e.g. *Matlab*, *Python*, *C/C++*) to solve an assignment, you must add the full sourcecode in the appendix of your PDF file and reference it in your description of the solution.

1 Follow the wall

In this lab, you will implement a PID (proportional integral derivative) controller to make the car drive parallel to the walls of a corridor at a fixed distance. At a high level, you will accomplish this by taking laser scan distances from the laser scanner (LiDAR), computing the required steering angle and speed (drive parameters), and publishing these to drive the car. Before starting this lab, review the lectures on PID and LiDAR to ensure you are familiar with the material.

Review of PID in the time domain

In the lecture we saw PID in the frequency domain, since that brings out most clearly the effects of the various gains and why we might want to add, say, a derivative term. Here we look at PID in the time domain, in relation to the task of wall-following.

A PID controller is a way to maintain certain parameters of a system around a specified set point. PID controllers are used in a variety of applications requiring closed-loop control, such as in the speed controller on your car.

The general equation for a PID controller in the time domain, as discussed in lecture, is as follows:

$$u(t) = K_p\, e(t) + K_i \int_0^t e(t')\, dt' + K_d \frac{d}{dt}(e(t))$$

Here, K_p, K_i, and K_d are constants that determine how much weight each of the three components (proportional, integral, derivative) contribute to the control output $u(t)$. $u(t)$ in our case is the steering angle we want the car to drive at. The error term $e(t)$ is the difference between the set point and the parameter we want to maintain around that set point.

Wall Following

In the context of our car, the desired distance to the wall should be our set point for our controller, which means our error is the difference between the desired and actual distance to the wall. This raises an important question: how do we measure the distance to the wall, and at what point in time? One option would simply be to consider the distance to the right wall at the current time t (let's call it D_t). Let's consider a generic orientation of the car with respect to the right wall and suppose the angle between the car's x-axis and the wall is denoted by α. We will obtain two laser scans (distances) to the wall: one at an angle θ ($0 < \theta \leq 70$ degrees), and another at an angle of 0 degrees relative to the car's x-axis. Suppose these two laser scans return distances a and b, respectively.

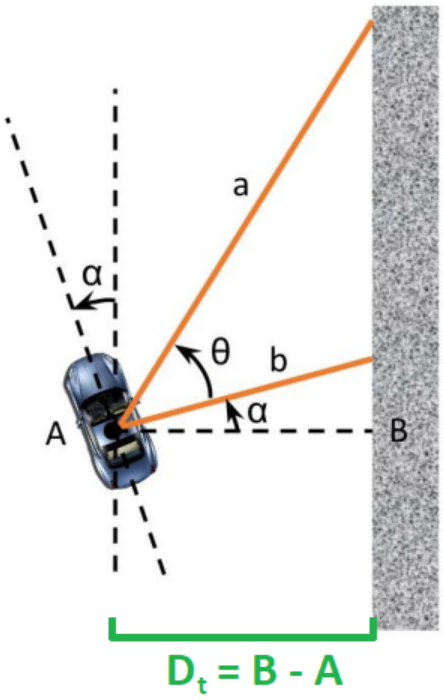

Figure 1: Distance and orientation of the car relative to the wall

Using the two distances a and b from the laser scan, the angle θ between the laser scans, and some trigonometry, we can express α as

$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta) - b}{a\sin(\theta)}\right) \tag{1.1}$$

We can then express D_t as

$$D_t = b\cos(\alpha)$$

to get the current distance between the car and the right wall. What's our error term $e(t)$, then? It's simply the difference between the desired distance and actual distance! For example, if our desired distance is 1 meter from the wall, then $e(t)$ becomes $1 - D_t$.

However, we have a problem on our hands. Remember that this is a race: your car will be traveling at a high speed and therefore will have a non-instantaneous response to whatever speed and servo control you give to it. If we simply use the current distance to the wall, we might end up turning too late, and the car may crash. Therefore, we must look to the future and project the car ahead by a certain lookahead distance (let's call it L). Our new distance D_{t+1} will then be

$$D_{t+1} = D_t + L\sin(\alpha)$$

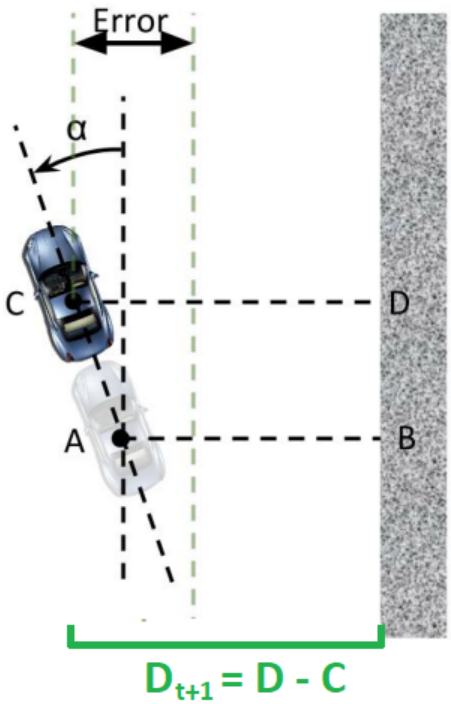

Figure 2: Finding the future distance from the car to the wall

We're almost there. Our control algorithm gives us a steering angle, but we would also like to slow the car down around corners for safety. We can compute the speed in a step-like fashion based on the steering angle so that as the angle exceeds progressively larger amounts, the speed is cut in discrete increments. For this lab, we would like you to implement the following speed control algorithm:

- If the steering angle is between 0 degrees and 10 degrees, the car should drive at 1.5 meters per second.
- If the steering angle is between 10 degrees and 20 degrees, the speed should be 1.0 meters per second.
- Otherwise, the speed should be 0.5 meters per second.

So, in summary, here's what we need to do:

1. Obtain two laser scans (distances) a and b, with b taken at 0 degrees and a taken at an angle θ $(0 < \theta \leq 70)$
2. Use the distances a and b to calculate the angle α between the car's x-axis and the right wall.
3. Use α to find the current distance D_t to the car, and then α and D_t to find the estimated future distance D_{t+1} to the wall.
4. Run D_{t+1} through the PID algorithm described above to get a steering angle.
5. Use the steering angle you computed in the previous step to compute a safe driving speed.
6. Publish the steering angle and driving speed to the VESC.

1.1 Implementation

Implement left-wall following to make the car drive autonomously. You can implement this node in either C++ or Python. However the optional skeleton code is only available in Python. The skeleton file is provided in the archive `exercise3_report_template_and_data.tar.bz2` in TUWEL.

a.) Create a new ROS package with name `wall_follow`.
Implement the wall following controller as described above to follow the **left** wall. The `AckermannDriveStamped` message should be published to topic `/nav`.
Test your node on the provided `f1_aut_wide` map [And22]. Add your source files in the appendix of your submission document.

b.) Place your car in the simulator on the part *Rectangle* of the `test` maps [And22] (s.t. the left wall is inside) and let it drive a complete round using your wall following node.
Create a plot for the following variables over time (e.g. by using `rqt_plot` [Sta22a]): D_{t+1}, steering angle, driving speed. (If interesting sections are not properly visible on the whole plot, add another plot for a shorter representative step-response-type section.)

c.) Complete task (1.1.b.) in the opposite direction (s.t. the left wall is outside). Record the same plot as in task (1.1.b.). Compare these two plots and discuss the differences (if there are any).

d.) Make the parameters for K_p, K_i, and K_d adjustable via `rqt_reconfigure` [Sta22b]. Compare and describe the differences for three differently tuned controllers by plots which are similar as in task (1.1.b.).

e.) Test your implementations from task (1.1.a.) locally on your system in the simulator. When you are confident that the node works properly, use the testbench in TUWEL (*Lab 3 Wall Following Node Testbench (wall_follow)*) to evaluate your implementation. For this you need to upload a ZIP-File of your node containing the following files (use exactly these names for directories and files).

Directory structure for your implementation:

```
CMakeLists.txt
package.xml
scripts[1]
    safety_node.py
src[2]
    safety_node.cpp
```

The name (in package.xml) for your package should be: `wall_follow`
The name (in CMakeLists.txt) for your project should be: `wall_follow`
If you have a python script use this shebang: `#!/usr/bin/env python3`
If you use C++ the name for your executable should be: `wall_follow`

The testbench system will take some time until the simulation is processed and the log files are available. It includes both whitebox-tests on known tracks and also a blackbox test on a unknown track. There is also a rendered video available for each sucessfully executed whitebox testcase. If there are any questions regarding the testbench system, please write a message in the TUWEL forum. Any feedback regarding improvements of the testbench system is also highly welcome!

f.) (**optional**) Since racing is (nearly) all about lap time, you may also tune the driving speed of your node (instead of fixed values 0.5, 1.0 and 1.5 as mentioned above). Make sure that your car does not crash into the wall. Use any algorithm to infer the desired driving speed based only on the `LaserScan` message (you must not use the knowledge of the absolute position and/or the map or any other messages). We will add a lap timer to the testbench system and publish the results. Teams which can complete the blackbox test on the unknown track and are at least 10% faster than the median of all the teams will get bonus points. The fastest team that completes the blackbox test will also get bonus points. (Lap times are only counted if the same submission does also pass all whitebox testcases.)

Note: Make sure to press `n` on your keyboard in the terminal window that launched the simulator. This will activate the navigation node.

[2]The folder `scripts` and the file `safety_node.py` must only be included if you decided to implement it in Python.
[3]The folder `src` and the file `safety_node.cpp` must only be included if you decided to implement it in C++.

2 Hardware Race Car

As already introcuded in the lectures, for any cyber physical system there is a sim-to-real transfer gap. In this task you will start working on the hardware race car and most likely already experience these effects.

2.1 Emergency Brake

Before driving autonomously you will need to demonstrate proper functionality of your emergency brake node on the race car.

a.) Make a copy of your implementation from task (1.1.a.), rename it to `slow_crash` and modify it the following way: The car should drive continously with constant speed $v = 0.1m/s$ and without steering. This node will be used for testing the emergency brake node.

b.) On the hardware race car, topic names are partially different from the simulator. Write a launch file, that starts the *Safety Node* (as implemented in Lab 2), and the node `slow_crash` and does proper topic remapping [Sta22d]. Use the following topics on the car:

- LIDAR: `/scan`
- Odometry from VESC: `/vesc/odom`
- *Safety Node* brake boolean: `/brake_bool`
- *Safety Node* ackermann-drive: `/vesc/low_level/ackermann_cmd_mux/input/safety`
- Joypad: `/vesc/joy`
- Driving node ackermann-drive:: `/vesc/high_level/ackermann_cmd_mux/input/nav_0`

Describe the mapping and your considerations. Add your launch files in the appendix of your submission document.

c.) Run the implementation using the launch file from task (2.1.b.) and the node from task (2.1.a.) on the actual hardware race car. Show to one of our tutors that it is able to stop the car safely before crashing into the barrier of the racetrack.

d.) In ROS coordinate transform relations can be described by Tf [Sta22c]. Visualize the Tf-tree of our robot in the simulatur using Rviz [Sta22e] and include this in the submission document. Which transforms are relevant for the sensors (LIDAR, IMU, etc.) on the hardware race car? Measure the actual position of these sensors on *car7* and adjust the respective file in the simulator to fit this car. Add the file in the appendix of the submission document. (This is relevant to minimize the sim-to-real transfer gap for further labs.)

2.2 First Autonomous Driving

In this task you will let the car drive autonomously for the first time.

a.) Make a similar launch file as in task (2.1.b.), that starts the *Safety Node*, and the node `wall_follow` from task (1.1.a.).

b.) Run the implementation using the launch file from task (2.2.a.) on the actual hardware race car. Show to one of our tutors that it is able drive properly on the racetrack. (Note: There is no need to run an extensively tuned version of your algorithm. Any version which works is fine in this subtask.)

c.) Describe in the protocol if the node from task (1.1.a.) worked without modification for task (2.2.b.). If not, elaborate on the needed modifications and/or parameter adjustments.

Note: Make sure to press and hold the button `LP` on the joypad controller to enable driving manually. Press and hold the button `RP` to enable driving from the `/nav` topic.

Appendix

Grading

The following points can be achieved for each task of this exercise sheet:

Exercise	Points
1.1.a.	21
1.1.b.	5
1.1.c.	5
1.1.d.	7
1.1.e.	12
Subtotal	50
2.1.a.	3
2.1.b.	3
2.1.c.	16
2.1.d.	5
2.2.a.	1
2.2.b.	16
2.2.c.	6
Subtotal	50
1.1.f.	5 [3] + 5 [4]
max. Bonus points[5]	10
Grand Total	100 (110 incl. Bonus points)

[4] Bonus points for teams which can complete the blackbox test on the unknown track and are at least 10% faster than the median of all the teams.

[5] Bonus points for the fastest team that completes the blackbox test on the unknown track.

[6] Please note that bonus points do only count for your grade, if your are already positive.

References

[And22] Andreas Brandstätter. *f1tenth_simulator/maps*. 2022. URL: https://github.com/CPS-TUWien/f1tenth_simulator/blob/main/maps/.

[Sta22a] Stanford Artificial Intelligence Laboratory et al. *Documentation: rqt_plot*. 2022. URL: http://wiki.ros.org/rqt_plot.

[Sta22b] Stanford Artificial Intelligence Laboratory et al. *Documentation: rqt_reconfigure*. 2022. URL: http://wiki.ros.org/rqt_reconfigure.

[Sta22c] Stanford Artificial Intelligence Laboratory et al. *ROS TF*. 2022. URL: http://wiki.ros.org/tf.

[Sta22d] Stanford Artificial Intelligence Laboratory et al. *ROS Topic Remapping*. 2022. URL: http://wiki.ros.org/roslaunch/XML/remap.

[Sta22e] Stanford Artificial Intelligence Laboratory et al. *Rviz TF*. 2022. URL: http://wiki.ros.org/rviz/DisplayTypes/TF.

Acknowledgments

This course is based on F1TENTH Autonomous Racing which has been developed by the Safe Autonomous Systems Lab at the University of Pennsylvania (Dr. Rahul Mangharam) and was published under CC-NC-SA 4.0 license.