TECHNISCHE UNIVERSITÄT WIEN

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

Lab 6: Pure Pursuit and Race Lines

2022-05-17

Preface

Read all the instructions below carefully before you start working on the assignment, and before you make a submission. All sources of material and ressources must be properly cited (this also includes datasheets).

- 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 2022-05-31 23:59.
- For this assignment there is no exercise interview. However if submissions are unclear, students might be invited for exercise interviews.

Learning outcomes

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

- Work with maps and (optimal) trajectories on given maps.
- Driving the car autonomously via pure pursuit.
- Tuning algorithms/controllers.
- Use live visualization of internal variables to better tune algorithms/controllers.

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 2022-05-31 23:59. Use the provided latex template and do not forget to fill in the parts marked with "TODO". The anonymous version of the lab report (all pages except the first one with personal data) will be shared with all the teams after the submission deadline. In any case the report must meet an adequate level for layout and readability that is appropriate for the academic context. It needs to be detailed enough, so that another team could reproduce your work without additional information.
- Submit a ZIP-Archive with all the relevant source code in TUWEL until the same deadline.

Transparency of Contribution

Describe in your submission protocol briefly how you worked together. How did you structure your work distribution and collaboration? Who contributed how much effort to which part of the work? (If one, or more team members, are not able to work on this assignment, you must also transparently state this here.)
(Please do not understand this preamble wrong to somehow exaggerate your contribution estimation: If you are working together well in your team, it should anyway be no problem to briefly describe how you worked together.)

1 Paths on maps

In this lab, you will implement a *Pure Pursuit* algorithm to follow a path. Given a map, you need first to create this path in a separate node.

Suggested algorithm

There are many methods how to create a path from a given map. The following steps provide an overview of one possible algorithm. However you are free to use any other method. (As long as you use only data from the `/map` topic for this.)

1. Subscribe the topic `/map` and preprocess the information in `data` into a 2D binary array (lets further denote it with `map_binary`). The `info.origin` from the `OccupancyGrid` message gives you the starting point p_s.
2. Given the point p_s use the flood fill algorithm [Wik21] on the `map_binary` to get the drivable area (`drivable_area`).
3. Assume the finish line L_f is perpendicular to p_s and ranges left and right until the ends of the `drivable_area`. (This will be the same for all test maps of this assignment. Also the car orientation at p_s will stay the same for all testcases.)
4. Erode the `drivable_area` by n_{safety} pixels to determine a save drivable area (`save_area`). This keeps you away from obstacles (e.g. borders of the track) while driving.
5. Calculate the distance $d_f(x, y)$ to the finish line L_f in backward direction for each pixel in `save_area` recursively.
6. Create a pixel-path starting at p_s by recursively choosing the next point from the neighbouring pixel with the smallest value for d_f.
7. Increase the sparsity of the pixel-path by selecting each n_{sparse}-th point for the resulting path.
8. Publish the resulting path in a appropriate data structure as ROS message on the `/path` topic to be used in the pure pursuit algorithm later on.

1.1 Implementation

Implement an algorithm to generate a path from a given map. You can implement this node in either C++ or Python. However the optional skeleton code is only available in Python. The skeleton file (`planner.py`) is provided in the archive `exercise6_data.tar.bz2` in TUWEL.

a.) Create a new ROS package with name `pure_pursuit`.
Implement the algorithm to generate a path as described above (or your own alternative algorithm). Test your node on the provided map `bunch_of_sticks` [And22]. Add your source files in the appendix of your submission document.

b.) Visualize the path that your algorithm is calculating.

c.) Add screenshots of the following maps [And22] with the rendered path in your PDF protocol:

- `bunch_of_sticks`
- `train_pile_of_blocks`
- `f1_aut`
- `f1_aut_wide`
- `f1_gbr`
- `f1_esp`

d.) Did you need to modify the value for n_{safety} in subtask (1.1.c.)? If yes, describe the needed modifications.

e.) Pick one of the paths generated in subtask (1.1.c.). Visually analyze where the path looks (nearly) optimal, and where it could be improved. (E.g. compared to a trajectory, that a human driver with knowledge of the map would take.) Mark the relevant sections in a figure in your protocol and give arguments of your analysis.

f.) Is there a difference of the shortest path and the optimal path w.r.t. fastest lap time? Give arguments and describe your reasoning.

g.) How could you formulate your characterization from subtask (1.1.e.) of "good" and "bad" parts of the path in a (more) formal (mathematical) way?

h.) Give an idea how to improve the algorithm used in (1.1.a.) in accordance to (1.1.g.).

2 Pure Pursuit

Before starting to implement the algorithm, recap the technical report by Coulter [Cou92], as presented in the lecture (This will follow after the section on particle filters.). Another optional resource that might be useful is the work by Snider [Sni09].

Implement the *Pure Pursuit* algorithm. You can implement this node in either C++ or Python. However the optional skeleton code is only available in Python. The skeleton file (`pure_pursuit.py`) in the archive `exercise6_data .tar.bz2` in TUWEL.

2.1 Implementation

a.) Add the implementation of the *Pure Pursuit* algorithm in a separate node to the package from task (1.1.a) and subscribe to the `/path` topic. For this you should use the path as derived in task (1.) and subscribe to the topic `/odom` and use the information `pose.pose` to get your ground-truth position[1] of the car. For this subtask you may choose a constant value for the speed. The `AckermannDriveStamped` message should be published to topic `/nav`.
Test your node on the provided map `bunch_of_sticks` [And22]. Add your source files in the appendix of your submission document.

b.) Visualize the *goal point* that your algorithm is calculating while driving along the path. You may have a look at *Marker* messages with type *Sphere* [Sta21] to show the point in *rViz*. We suggest to use the topic `/marker_goal` for this.
Also visualize the path as in subtask (1.1.b.) simultaneously. Record a short video (approx. 15 seconds) of your car driving on a map of your choice to demonstrate the visualization and upload it in TUWEL at the separate submission item.

c.) For every n_{log}-th[2] timestep your node should output [Sta22] the ground-truth position (x- and y-coordinate) or optionally log it into a file.

d.) Pick one of the maps listed in subtask (1.1.c.), run your *Pure Pursuit* algorithm and plot the trajectory of the car (coordinates from subtask (2.1.c.)) for at least two different values for the lookahead distance L.

e.) Discuss the influence of the lookahead distance L as seen in subtask (2.1.d.) for your particular implementation.

f.) Test your implementation from subtask (2.1.a.) locally on your system in the simulator. When you are confident that the node works properly, use the testbench in TUWEL (*Lab 6 Pure Pursuit Testbench (pure_pursuit)*) 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).

[1] For this sub-task we are using ground-truth positions to simplify this for now. This is of course only directly available in simulation.
[2] You may choose an appropriate value for n_{log}.

Directory structure for your implementation:

```
CMakeLists.txt
package.xml
scripts[3]
    planner.py
    pure_pursuit.py
src[4]
    pure_pursuit.cpp
    planner.cpp
```

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

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 successfully 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!
The relevant testcases for this subtask are `testcase0: bunch_of_sticks` and `testcase1: f1_aut`. Your implementation must be able to pass these two testcases without crash in the given time limit.

2.2 Building an more advanced controller

Based on the simple implementation you should now tune or improve your controller.

a.) (**alternative 1**) Tune your controller for faster driving speed. Adjust your controller for variable driving speed of your node (instead of just a fixed value as mentioned above). Make sure that your car does not crash into the wall. Use any algorithm to infer the desired driving speed. In this lab you may use information from `LaserScan` messages and also the knowledge of the absolute position and/or the map (but you must not use the velocity information in the messages of the `/odom` topic.). Describe the implemented solution in detail in your submission protocol:

- Describe how you came up with your solution method and what's the idea behind it.
- Use appropriate graphs/tables/etc. to support your description.

b.) (**alternative 2**) Optimize your race line. Modify your path planning to get a better race line. As the shortest path may not be the fastest one to follow, it is needed to adjust the path generation accordingly. Describe the implemented solution in detail in your submission protocol:

- Describe how you came up with your solution method and what's the idea behind it.
- Use appropriate graphs/tables/etc. to support your description.
- Give arguments why your path is faster than the shortest path.

c.) (**alternative 3**) Avoid obstacles. As the *Pure Pursuit* algorithm does not incorporate lidar data, it is not able to avoid obstacles that are not on the map. Design a modified version of the algorithm that is able to detect if the next point of the path is safely reachable. If this is not the case, it should react accordingly. (One strategy might be to slow down or to stop the car. This is simplified reaction is fine for now in this sub-task.) Describe the implemented solution in detail in your submission protocol:

- Describe how you came up with your solution method and what's the idea behind it.
- Use appropriate graphs/tables/etc. to support your description.
- Describe how you tested your implementation in the simulator.

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

3 Proposal for further work

As until now you have implemented given methods and algorithms to get familiar with the environment of autonomous racing. You are now familiar with the capabilities of the cars and should have an idea what challanges still need to be solved. In the last two labs you will have more flexibility to agree within your team what plan you want to follow in order to get your car driving fast, safe and reliable.

a.) This subtask is to give an abstract (approx. 1 page of text) of what further plan you would like to follow in the next two labs. This might include:

- If you want to fist focus on a specific sub-task (e.g. behaviour in specific scenarios, improvement of relevant metrics, some type of specific optimization). But it is not needed to choose a specific sub-task. You can also focus on racing in general.
- How you want to approach your task.
- If you want to build on stuff that you (or other teams) already implemented in previous labs, or other work that is published somewhere else.
- How you want to assess if your goal was reached (which metrics to use, etc.).
- Potential breakdown of the work in smaller sub-packages.
- Any other relevant information.

We will give you feedback on this abstract and after clearance (and potential modification by us), you can work on this task for Lab 7.

Appendix

Grading

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

Exercise	Points
1.1.a.	10
1.1.b.	3
1.1.c.	6
1.1.d.	2
1.1.e.	4
1.1.f.	4
1.1.g.	4
1.1.h.	6
Subtotal	39
2.1.a.	15
2.1.b.	4
2.1.c.	4
2.1.d.	4
2.1.e.	5
2.1.f.	9
Subtotal	41
2.2.a.	10 [5]
2.2.b.	10 [5]
2.2.c.	10 [5]
Subtotal	10
3.a.	10
Subtotal	10
Grand Total	100

[5] Implement one of these sub-task.

References

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

[Cou92] R. Craig Coulter. *Implementation of the Pure Pursuit Path Tracking Algorithm*. 1992. URL: https://www.ri.cmu.edu/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf.

[Sni09] Jarrod M. Snider. *Automatic Steering Methods for Autonomous Automobile Path Tracking*. 2009. URL: https://www.ri.cmu.edu/pub_files/2009/2/Automatic_Steering_Methods_for_Autonomous_Automobile_Path_Tracking.pdf.

[Sta21] Stanford Artificial Intelligence Laboratory et al. *DisplayTypes/Marker*. 2021. URL: http://wiki.ros.org/rviz/DisplayTypes/Marker.

[Sta22] Stanford Artificial Intelligence Laboratory et al. *ROS Logging*. 2022. URL: http://wiki.ros.org/roscpp/Overview/Logging.

[Wik21] Wikipedia. *Flood fill — Wikipedia, The Free Encyclopedia*. [Online; accessed 2021-04-28]. 2021. URL: https://en.wikipedia.org/wiki/Flood_fill.

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.