TECHNISCHE UNIVERSITÄT WIEN

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

Lab 5: Reactive Methods

2022-05-03

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 2022-05-17 23:59.
- For this assignment there is no exercise interview. However if submissions are unclear, students might be invited for exercise interviews.
- One team will additionally present their work at the *Lab 5 presentation*.

Learning outcomes

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

- Work with reactive methods for obstacle avoidance.
- Driving the car autonomously via a reactive algorithmn.
- Tuning algorithmns/controllers.
- Use live visualization of internal variables to better tune algorithmns/controllers.

Deliverables and Submission

- Write a exercise report and submit it as PDF file in TUWEL until 2022-05-17 23:59. Submit a video file in TUWEL until the same deadline. 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 Reactive algorithmns

In this lab, you will implement a reactive algorithm for driving and obstacle avoidance. While we suggest to use *Follow the Gap* or *Disparity extender*, you are encouraged to try different reactive algorithms or a combination of several to improve your solution.

Follow the Gap

The lecture slides on *Follow the Gap* is the best visual resource for understanding every step of the algorithm. However, the steps are outlined over here:

1. Obtain laser scans and preprocess them
2. Find the closest point in the LiDAR ranges array
3. Draw a safety bubble around this closest point and set all points inside this bubble to 0. All other non-zero points are now considered "gaps" or "free space"
4. Find the max length "gap", in other words, the largest number of consecutive non-zero elements in your ranges array
5. Find the best goal point in this gap. Naively, this could be the furthest point away in your gap, but you can probably go faster if you follow the "Better Idea" method as described in lecture.
6. Actuate the car to move towards this goal point by publishing an `AckermannDriveStamped` to the `/nav` topic

Disparity Extender

The alogorithm called *Disparity Extender* [Nat19] is another reactive algorithmn which was winning the F1/Tenth competition, held at CPSWeek 2019, in Montreal. The main steps (refer to *Disparity Extender*) are outlined here:

1. Acquire the LIDAR data
2. Find the "disparities" in the LIDAR data
3. For each disparity
   (a) Find the nearer of the two points, and calculate the number of LIDAR samples neccessary to "mask out" the region of samples unsafely close to the obstacle at that distance.
   (b) Overwrite the number of samples calculated previously with the shorter distance, starting at the disparity, and moving in the direction of the longer distances. (Don't overwrite distances that are already closer!)
4. Search the new "filtered" distances for the farthest point that isn't behind the car. Calculate the angle that this point is in.
5. Steer in the angle you calculated (or as much towards it as possible). Make sure that there are no obstacles on the side of the car that you'll hit due to turning; go straight instead if there are. Set the speed of the car based on the distance directly in front of the car. Publish an `AckermannDriveStamped` to the correct topic.

1.1 Implementation

Implement one reactive algorithm to make the car drive autonomously. You can implement this node in either C++ or Python. If you want, you can use some basic skeleton file from a previous lab.

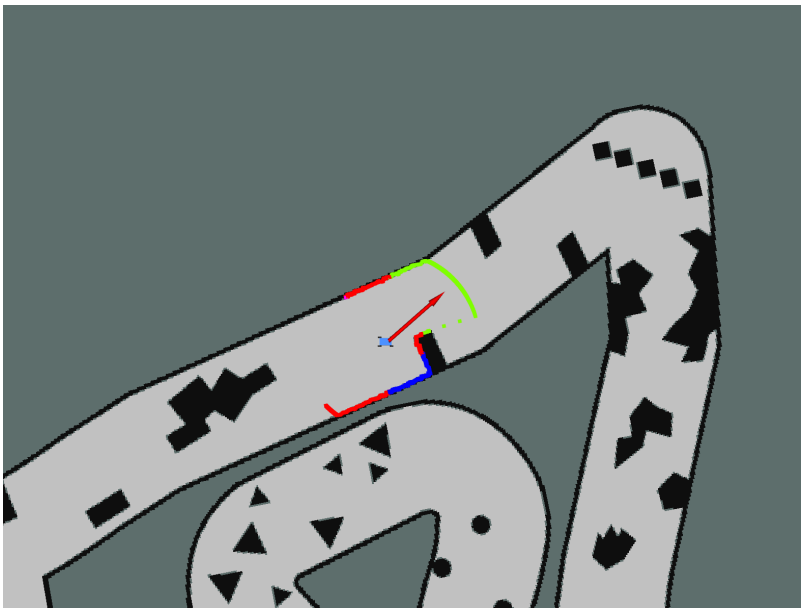

Figure 1: Example how the *threshold*, the gaps and the driving command can be visualized for *Follow the Gap*. The gaps are shown in blue, while the best gap is shown in green. The driving command (direction and speed) is indicated with an arrow.

a.) Create a new ROS package with name `follow_the_gap`.
Implement a reactive controller as described above. 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 `f1_aut_wide` map [And22]. Add your source files in the appendix of your submission document.

b.) Visualize the internals of the controller (for *Follow the Gap* this would be *threshold* and the gaps) and the driving command (direction and speed) that your algorithm is calculating. (Refer to figure 1 for an example how this visualization could look like.) You may have a look at *Marker* messages **DisplayTypesMarker** and/or publish *LaserScan* messages on a custom topic with modified information in the field `intensities` to show different colors in *rViz*.
Record a short video (approx. 15 seconds) of your car driving on a map of your choice (for example `f1_aut_wide_obstacles` [And22]) to demonstrate the visualization and upload it in TUWEL at the separete submission item.

c.) Run your car in the simulator on the part *Rectangle* of the `test` maps [And22] and on the map with obstacles: `f1_aut_wide_obstacles`. Adjust the relevant internals of the controller. Discuss the influence of these value(s) in your report:

- Describe the different step-response behaviour (as seen on the *Rectangle* part).
- Describe the different behaviour in front of obstacles.
- Use appropriate graphs/tables/etc. to support your arguments.

(For this subtask it is ok, if the controller crashes into the wall on the `f1_aut_wide_obstacles` map. Depending on the type of reactive algorithm, there might be more or less options for adjusting internal values. Also the influence might vary.)

d.) Test your implementation from subtask (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 4 Reactive Methods Testbench (reactive)*) 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]
    reactive.py
src[2]
    reactive.cpp
```

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

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!
The relevant testcases for this subtask are `testcase0: f1_aut_wide` and `testcase1: f1_aut_wide reversed` ↪ . Your implementation must be able to pass these two testcases without crash in the given timelimit.

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

e.) Run your implementation from subtask (1.1.a.) on the hardware race car. Show to our tutors that it is properly able to drive on a test-track at TU Wien. For this subtask it is sufficient to use a version of your controller that is not tuned to be fast. The main goal is to drive autonomously without crashing.

f.) Where there modifications (e.g. of parameters) needed to adjust your solution of task (1.1.d.) to the task (1.1.e.)? If yes, give details and possible reasons.

1.2 Tuning the controller

Based on the simple implementation you should now tune your controller. There are two different objectives (which might partially contradict each other): One is to drive really fast on (easy) maps. The other is to be able to also complete difficult tracks without crashing.
To limit the workload for this task, it is enough if you solve one of the subtasks (1.2.a.) or (1.2.b.) or (1.2.c.). This gives you the freedom to choose what objective you want to tackle.

a.) (**alternative 1**) Tune your controller so that it successfully completes the following testcases: `testcase2 : f1_aut_wide_obstacles` and `testcase3: f1_aut_wide_obstacles reversed`. Your implementation must be able to pass these two testcases without crash in the given timelimit (while still passing `testcase0` and `testcase1`). Refer to subtask (1.1.d.) for details about the testbench system.
Describe the implemented solution in detail in your submission protocol:

- Discuss (at least two) different attempts and compare them.
- Use appropriate graphs/tables/etc. to support your arguments.
- Describe how you came up with your solution method and whats the idea behind it.

b.) (**alternative 2**) 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 based on the `LaserScan` message (you must not use the knowledge of the absolute position and/or the map). The relevant testcases for this subtask are `testcase0: f1_aut_wide` and `testcase1: f1_aut_wide reversed`. Refer to subtask (1.1.d.) for details about the testbench system.
Describe the implemented solution in detail in your submission protocol:

- Discuss (at least two) different attempts and compare them.
- Use appropriate graphs/tables/etc. to support your arguments.
- Describe how you came up with your solution method and whats the idea behind it.

c.) (**alternative 3**) Tune your controller for faster driving on the real hardware. This is similar to (1.2.b.) but evaluation must be done on the real hardware racecar instead of simulation. For this you should use the content of Lab 4 to localize your car on the test-track at TU Wien. Record traces of your localized pose to assess the effects of your tuning efforts (e.g. with respect to optimal trajectory and lap-times). Describe the implemented solution in detail in your submission protocol:

- Discuss (at least two) different attempts and compare them.
- Give multiple plots/pictures of the recorded traces on the test-track.
- Use appropriate graphs/tables/etc. to support your arguments.
- Describe how you came up with your solution method and whats the idea behind it.

d.) (**optional**) There is a lap timer in the testbench system which results are published. 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. Please note for this task we do only count lap times if all the other testcases are also passed in the same execution run. I.e. all green checkmarks. (We will evaluate this manually after the deadline.)
Yes, we know that this might be very challenging, since you need to complete the `f1_aut_wide_obstacles` map and be fast at the same time. This is, why this subtask is optional.

Notes

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

Appendix

Grading

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

Exercise	Points
1.1.a.	20
1.1.b.	12
1.1.c.	10
1.1.d.	8
1.1.e.	20
1.1.f.	5
Subtotal	75
1.2.a.	25 [3]
1.2.b.	25 [3]
1.2.c.	25 [3] + 8 [4]
1.2.d.	4 [5] + 4 [6]
Subtotal	25
Grand Total	100 (excl. Bonus points [7])

[3] You can solve one of the subtasks (1.2.a.) or (1.2.b.) or (1.2.c.).

[5] Bonus points for teams which achieve really good results for tuning the controller on the hardware race car.

[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.

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

[7] 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/.

[Nat19] Nathan Otterness et al. *The Disparity Extender Algorithm, and F1/Tenth*. Apr. 22, 2019. URL: https://www.nathanotterness.com/2019/04/the-disparity-extender-algorithm-and.html.

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.