HAW HAMBURG

Fakultät TI, Department Informatik
Intelligente Sensorik
Prof. Dr. Tim Tiedemann, Max De Muirier

Übungsblatt 3
zur Veranstaltung „AFR – Autonomes Fahren und Robotik”

3. Programmierung

In der Übung 3 geht es um Navigationsstrategien und Pfadplanung innerhalb bekannter und bereits kartierter Umgebungen.

3.1. Vorbereitungsaufgaben

1. Schauen Sie sich einmal die Beschreibungen zu den **ROS**-packages *AMCL*, *teb_local_planner*, *dwa_local_planner* und *global_planner* im *ROS-Wiki* an. Was wird zu den Algorithmen und zur Funktionsweise gesagt?

3.2. Praktikumsaufgaben

1. Bearbeiten Sie die Schritte **Localization and navigation within known environments** (Anhang A). Wie findest sich das Fahrzeug in der Umgebung zurecht? Beschreiben Sie, was nach der Positionsschätzung passiert. *Protokoll!*

2. Ändern Sie den *Global planner* von **D*** zu **A*** mittels *use_quadratic*. Wie beeinflusst diese Änderung den Pfadplanungsprozess? Probieren Sie die anderen beiden *Local planner*. Wie unterscheiden diese sich voneinander? Optimieren Sie die Parameter. *Protokoll!*

3. Welche Aufgabe haben die lokale und globale Kostenkarte? Verändern Sie diese mittels *inflation_radius* und *cost_scaling_factor*. Wie wirkt sich diese Parameter aus? *Protokoll!*

4. Bearbeiten Sie die Schritte **Using the build-in navigation in Carla Simulator** (Anhang B). Beschreiben Sie den Ablauf der Pfadplanung und Pfadverfolgung im Carla Simulator. Welche Komponenten sind in **RVIZ** zu sehen? *Protokoll!*

5. Betrachten Sie die entsprechende *launch file* unter *carla_ad_demo/launch* zur AD Demo. Welche Prozesse werden aufgerufen? Überlegen Sie, welche Aufgaben diese erfüllen. *Protokoll!*

6. Ändern Sie den *spawn_point* ab und durchlaufen Sie erneut die Demo. Die Werte entnehmen Sie dem Pygame-Fenster an einer Position ihrer Wahl.

7. Schauen Sie sich das dazugehörige Python-Skript zur Pfadplanung unter *carla_waypoint_publisher/src/carla_waypoint_publisher* an. Wie wird hier die Pfadplanung realisiert? Welche Informationen erhält das AD von der Simulation? *Protokoll!*

8. Wie unterscheiden sich die Aufgaben des *Global planner* und *Local planner*? Wie ist der *Local planner* unter *carla_ad_agent/src/carla_ad_agent* im Carla Simulator implementiert? Wie wird hier die Pfadverfolgung geregelt? *Protokoll!*

9. Bearbeiten Sie die Schritte **Path planning and tracking** (Anhang C). Beschreiben Sie das Verfahren *Pure Pursuit*. Schauen Sie dazu auch in den dazugehörigen Code. *Protokoll!*

10. Optional: Setzen Sie sich mit dem Code Beispiel auseinander. Wie ist hier die die *Pfadplanung* und *-verfolgung* implementiert.

A. Localization and navigation within known environments

In the previous session we have learned how to generate an occupancy grid map of an unknown environment in the **Carla**. Now we want to localize ourself within that map and navigate.

In this session we will make use of *Adaptive Monte Carla Localization* and the build-in features of the **Carla**. All required files for this workshop can be found in the ROS package *afrp3* and *ros-bridge*. Just download them from **EMIL** and move it into the *source* directory of your *catkin workspace*.

Append the *.bashrc* from the **ROS**-package *afrp3* to your *.bashrc* and source it.

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
echo "source ~/carla_ws/src/afrp3/.bashrc" >> ~/.bashrc
source ~/.bashrc
```

Localization in grid-based maps with AMCL

Let's start **Carla**, the **Carla ROS Bridge** and **RVIZ**.

```
/opt/carla-simulator/CarlaUE4.sh --quality-level=Low &
roscore &
source ~/carla_ws/devel/setup.bash
roslaunch afrp3 carla_ros_bridge.launch
roslaunch afrp3 carla_rviz.launch
```

Our vehicle named *ego_vehicle* has spawned. Now we need to convert the generated *3D point cloud* to a *2D laser scan* and all *Twist* commands to *Carla Control* commands:

```
roslaunch afrp3 carla_laser_scan.launch
roslaunch carla_twist_to_control carla_twist_to_control.launch
```

Load your previously generated map or choose one of the two provided:

```
roslaunch afrp3 carla_map.launch map_type:=hector or gmapping
```

Start *AMCL*:

```
roslaunch afrp3 carla_amcl.launch
```

Since *AMCL* doesn't perform a initial position estimation and just spawns the vehicle at the position predefined in the *launch* file, **2D Pose Estimate** can set a starting point. It can also be used to reposition the vehicle at any given time.

The **RQT Gui** can be used to control the vehicle using *Twist* commands.

```
rosrun rqt_robot_steering rqt_robot_steering
```

Now, in the upper left corner, change the subscribed topic to */carla/ego_vehicle/twist*. And set the maximum/minimum velocity to *10.0/-10.0*. Steering should now work.

Path planning and tracking with move_base

To autonomously navigate within the known environment, we need a *global planner* and *local planner* with their respective costmaps. The global path planning the algorithms *A** and *D** are available. For local path planning the packages *base_planner*, *DWA_local_planner* and *TEB_local_planner* can be used. The costmaps will help to find the optimal path to the target position while maintaining the lowest costs.

By default, *A** is set for global planning. Set *DWA* for local planning. Now, let's start the navigation:

```
roslaunch afrp3 carla_move_base.launch dwa:=true
```

Use **2D Nav Goal** to set a target position.

Editing parameters

All parameters within **ROS** can be seen and modified in real-time via **RQT**:

```
rosrun rqt_gui rqt_gui
```

Note: This will only temporarily change the settings! But the effects of the change will show immediately!

Parameter settings for the *global* and *local costmaps* can be found in:

```
cd ~/carla_ws/src/afrp3/param
gedit global_costmap_params.yaml
gedit local_costmap_params.yaml
```

The *inflation_radius* and *cost_scaling_factor* will have a significant influence on the costmaps and thus on the path planning process.

To choose between the *A** and *D** algorithm for the global navigation, edit the following param file:

```
gedit global_planner_params.yaml
```

The local planner can be changes by setting the arguments *dwa*, *teb* or *base* to *true*. The respective parameter file will be loaded as well:

```
roslaunch afrp3 carla_move_base.launch dwa:=true or teb:=true or
base:=true
```

Start **RQT** to change parameters:

```
rosrun rqt_reconfigure rqt_reconfigure --force-discover
```

and confirm your changes with **ENTER**. Hit **Refresh** in case parameters are missing.

Note: Changes parameters will not be saved!

The commands for localization with **AMCL** and navigation with **move_base** can be started simultaneously using the provided launch file *start_carla_navigation.launch*.

Just type:

```
roslaunch afrp3 start_carla_navigation.launch slam_method:=hector
   or gmapping dwa:=true
```

B. Using the build-in navigation in Carla Simulator

The Autonomous Driving Demo

Start the **Carla Simulator** with your graphic settings, the **roscore** and **RVIZ**:

```
/opt/carla-simulator/CarlaUE4.sh -quality-level=Low &
roscore &
rosrun rviz rviz -d ~/carla_ws/src/afrp3/rviz/carla_ad_demo.rviz &
```

Start the demo:

```
roslaunch carla_ad_demo carla_ad_demo.launch
```

Now, the generated path as well as the current and target pose of the vehicle can be seen. Throughout the path tracking process, both are updated.

The respective *launch file* can be viewed and edited in:

```
gedit ~/carla_ws/src/ros-bridge/carla_ad_demo/launch/carla_ad_demo
   .launch
```

To take a deeper look into how the path planning is implemented in this demo, the respective python script has to be viewed in:

```
gedit ~/carla_ws/src/ros-bridge/carla_waypoint_publisher/src/carla
   _waypoint_publisher/carla_waypoint_publisher.py
```

C. Path planning and tracking

As you have seen in the previous chapters, the **Carla** offers the possibility to interact with its simulation environment via a *PythonAPI*. Examples are at:

```
cd /opt/carla-simulator/PythonAPI/examples
```

A full documentation of all features can be found at:

`https://carla.readthedocs.io/en/latest/python_api/`

For more tutorials, visit:

`https://carla.readthedocs.io/en/0.9.2/python_api_tutorial/`

Note: Some tutorials may be outdated!

Lateral and longitudinal control for path tracking

The following algorithms for *lateral control* or “steering” of the vehicle in *path tracking* are commonly used in Autonomous Driving:

1. Pure Pursuit
2. Model predictive control (MPC)
3. Linear-quadratic regulator (LQR)
4. Stanley Controller (Stanford University)

A PID-controller is used for *longitudinal control*.

This workshop contains a small demonstration that shows how the *Pure Pursuit*-Algorithm works.

To start the very simplified simulation, go to the *Pure Pursuit example* in *examples* folder and start the simulation:

```
cd ~/catkin_ws/src/afrp3/examples/pure_pursuit
python sim.py
```

Navigation specific python scripts, that are implemented in *Carla Simulator* can be found in:

```
cd /opt/carla-simulator/PythonAPI/carla/agents/navigation
```

The lateral and longitudinal controller is implemented in *controller.py*. A closer look into these files will help to understand how the general navigation is implemented.

Code Example

This seesion also includes a basic code example for navigation using the **PythonAPI**. That includes python scripts for setup of the simulation environment, the vehicle and the camera.

Start the simulation (not working with Carla 0.9.13):

```
cd ~/carla_ws/src/afrp3/examples/carla_pure_pursuit
python citymap.py
python camera.py
python simpleclient.py
```

The original version can be found at: `https://github.com/copotron/sdv-course`
A working example can be found:

```
cd ~/carla_ws/src/afrp3/examples/carla_navigation
python carla_example.py
```

D. Higher functions of Autonomous Driving in Carla Simulator

Carla has developed a **scenario runner** to simulate predefined traffic scenarios.

Just download the latest *Scenario Runner* from **github** to your *catkin source* folder and build the entire folder again.

```
git clone https://github.com/carla-simulator/scenario_runner
cd ~/carla_ws
rm -rf build/ devel/
catkin_make
```

To start, go to the scenario directory, set the scenario and start manual control:

```
cd src/scenario_runner
./scenario_runner.py --scenario FollowLeadingVehicle_1 --reloadWorld
./manual_control.py
```

A list of scenarios can be found here:
`https://carla-scenariorunner.readthedocs.io/en/latest/list_of_scenarios/`

Full documentation can be found here:
`https://carla-scenariorunner.readthedocs.io/en/latest/`