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

# Übungsblatt 2
# zur Veranstaltung „AFR – Autonomes Fahren und Robotik"

## 2. Simultaneous Localization and Mapping

In der Übung 2 geht es um Algorithmen zum Aufbau von und Lokalisation in Umgebungskarten. Untersucht werden die Verfahren GMapping und HectorSLAM.

**Zur Bearbeitung der Übungsaufgaben:** Bearbeiten Sie auf jeden Fall alle Übungsaufgaben. Ausgenommen hiervon sind lediglich die mit „optional" gekennzeichneten Textstellen. Lesen Sie sich die Aufgaben gut durch. Sollten Sie eine Aufgabe nicht lösen können, so beschreiben Sie zumindest, wie weit Sie gekommen sind und auf welche Weise Sie vorgegangen sind. Die Aufgaben sind direkt im Protokoll zu beantworten. Als Lösung der Aufgaben ist alternativ Programmcode abzugeben (kommentiert) oder ein kurzer Text.

### 2.1. Vorbereitungsaufgaben

1. Bereiten Sie sich auf die Anwendung von GMapping und HectorSLAM vor, in dem Sie sich die in der Vorlesung genannten Papers bzw. Buchauszüge anschauen. Wie funktionieren die beiden Algorithmen?

2. PID-Regler: Inverses Pendel

   a) Bearbeiten Sie die auf den folgenden Seiten beschriebenen Schritten zu **PID Control Example 1: Inverse Pendulum** (Anhang C).

   b) Wie ist das Experiment aufgebaut und was ist das Ziel dieser Regelung? Worin genau liegt die Schwierigkeit? *Protokoll!*

   c) Welche Komponenten des Experimentes werden wie geregelt? Schauen Sie dazu in den Quellcode des Python-Skriptes *pid_command.py*. *Protokoll!*

   d) Schauen Sie sich den Verlauf der Parameter in **RQT** an. Applizieren Sie nun eine geringe Kraft von 30 N auf *tip_link*. Was ist zu beobachten? *Protokoll!*

   e) Wiederholen Sie die vorherigen Schritt bis die Regelung nicht mehr nachkommt. Welches Kompensationsverhalten ist nun zu beobachten? *Protokoll!*

   f) Ändern Sie nun die Parameter der Regelung. Wie wirken sich die Änderungen einzelner Parameter auf die Regelung aus?

### 2.2. Praktikumsaufgaben

1. Bearbeiten Sie die Schritte in **Running Carla Simulator and Carla ROS Bridge** (Kapitel A). Welche Sensordaten werden von der **ROS Bridge** generiert? Nennen Sie die entsprechenden Namen der Topics und der jeweiligen Nachrichtentypen. *Protokoll!*

2. Öffen Sie nun **RViz** und schauen Sie sich die verschiedenen Sensordaten an (fügen Sie Ihrem Protokoll ggf. Screenshots hinzu). Welche Art Sensoren finden Anwendung und wie unterscheiden sich die Sensordaten voneinander? Erklären sie, warum so viele unterschiedliche Sensoren verwendet werden! *Protokoll!*

3. Erstellen Sie eine Übersicht aller *nodes*, *topics* und *tfs* mittels **RQT**. *Protokoll!*

4. Wo lassen sich die Parameter der simulierten Sensoren betrachten und verändern?

5. Welche Sensordaten werden für die Lokalisation und Kartierung benötigt? *Protokoll!*

6. Starten Sie das Verfahren **HectorSLAM**. Welche Sensordaten werden für dieses Verfahren benötigt? Welche Aufgabe hat der Node *pointcloud_to_laserscan*? Fahren Sie eine kleine Runde um den Block und erstellen Sie eine Karte. Speichern Sie diese ab. *Protokoll!*

7. Erhöhen Sie nun die Genauigkeit des Verfahrens auf *0.025* (resolution). Betrachten Sie die erzeugte Karte der Umgebung. Nun verringern Sie sie auf *0.3*. Was ist zu beobachten? Welche Vor- und Nachteile hat eine höhere Genauigkeit? *Protokoll!*

8. Setzen Sie nun den Parameter *map_multi_res_levels* auf *5*. Erzeugen Sie eine Karte. Was ist bei dem Kartierungsprozess zu beobachten? Setzen Sie nun den Parameter auf *1*. Erzeugen Sie auch hier eine Karte und vergleiche Sie beide miteinander. Erklären Sie, welche Vor- und Nachteile dieser Parameter hat? *Protokoll!*

9. Starten Sie nun das Verfahren **GMapping**. Welche Sensordaten werden für dieses Verfahren benötigt? Fahren Sie um den Block und erzeugen Sie eine Karte. Wie unterscheidet sich dieses Verfahren und dessen Performance zum Verfahren HectorSLAM? *Protokoll!*

10. Was sind Partikel und welche Aufgabe haben sie? Welche Rolle spielt die Odometrie? Könnte man auch nur die Odometrie für die Lokalisation nutzen?

11. Erhöhen Sie nun die Partikelzahl auf *100*. Was ist zu beobachten? Welche Vor- und Nachteil bringt diese Änderung? Welchen großen Nachteil hat es, wenn man nur einen Partikel verwenden würde? *Protokoll!*

12. Welchen Einfluss hat die Fahrt- und Drehgeschwindigkeit auf den Kartierungsprozess? Warum ist das so? Wo liegt der Flaschenhals?

13. Ändern Sie nun die Sensorreichweite (*range*) zu den Einträgen *lidar* und *lidar_raycast* in *objects.json* sowie in den launch-files zu *gmapping* und *carla_laser_scan* auf *100*. Welchen Einfluss hat diese Änderung auf den Kartierungsprozess? Welche Vor- und Nachteile hat die Erhöhung der Sensoreichweite? *Protokoll!*

14. **Optional:** Informieren Sie sich über das Verfahren RTABmap. Bearbeiten Sie die (optionalen) Schritte zu diesem Verfahren.

# A. Running Carla Simulator and Carla ROS Bridge

Create a workspace and clone the *ROS Bridge* repository:

```
mkdir -p ~/carla_ws/src
cd ~/carla_ws/src
git clone https://github.com/carla-simulator/ros-bridge.git
```

Source the right path to **ROS**, set up its environment and install necessary dependencies:

```
source /opt/ros/noetic/setup.bash
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
cd ros-bridge
```

Before building the workspace, we need to install **Python** and **ROS** dependencies:

```
sudo apt-get update
python -m pip install -r requirements.txt
rosdep update
rosdep install --from-paths src --ignore-src --rosdistro noetic -y
```

You can now build **Carla ROS Bridge**:

```
cd ~/carla_ws
catkin_make
```

Lastly, set the right path to the **Carla** python modules and to the respective workspace:

```
source ~/carla_ws/devel/setup.bash
echo "export CARLA_ROOT=/opt/carla-simulator/" >> ~/.bashrc
echo "export PYTHONPATH=$PYTHONPATH:$CARLA_ROOT/PythonAPI/carla/
   dist/carla-0.9.13-py3.7-linux-x86_64.egg:$CARLA_ROOT/PythonAPI/
   carla" >> ~/.bashrc
echo "source ~/carla_ws/devel/setup.bash" >> ~/.bashrc
```

Once the setup is completed, source the **ROS** environment (if not yet set in *~/.bashrc* and run **roscore**. Then follow with starting the **Carla** server and the **Carla ROS Bridge**:

```
/opt/carla-simulator/CarlaUE4.sh
roslaunch carla_ros_bridge
   carla_ros_bridge_with_example_ego_vehicle.launch
```

Press *B* to switch to manual control and then steer with *W, A, S, D*.

## B. LiDAR-based SLAM Algorithms for Carla Simulator

Download and unzip the **ROS**-package **afrp2** to the *source* folder of your *catkin workspace*. This package includes all the files that you'll need for this part of the course.

### Running SLAM in Carla Simulator

To sum up how to start the **Carla** with the **Carla ROS Bridge**:

```
19 /opt/carla-simulator/CarlaUE4.sh &
20 source /opt/ros/noetic/setup.bash
21 source  /carla_ws/devel/setup.bash
22 roscore &
23 roslaunch carla_ros_bridge carla_ros_bridge_with_example_ego_vehicle
   .launch
```

**Note:** In case of bad performance, the simulator can be started in low resolution mode with the argument *-quality_level=Low*.

To start *HectorSLAM*, close every other **Carla** client and follow with:

```
24 roslaunch afrp2 start_hector_slam.launch
```

To start *GMapping*, follow with:

```
25 roslaunch afrp2 start_gmapping.launch
```

**Optional:** To start *RTABmap*, follow with: (Not working properly)

```
26 roslaunch afrp2 carla_rtabmap.launch
```

A generated map can be saved under a given name by typing:

```
27 rosrun map_server map_saver -f ~/carla_ws/map_name
```

and displayed with:

```
28 rosrun map_server map_server ~/carla_ws/map_name.yaml
```

Settings for each *SLAM* algorithm can be modified within the respective *launch* files, e.g., for *HectorSLAM*:

```
29 gedit ~/carla_ws/src/afrp2/launch/start_hector_slam.launch
```

## C. PID Control Example 1: Inverse Pendulum

Download the **ROS** package *cart_pole* from **Teams** and unzip it to your *home* directory:

```
30 unzip ~/Downloads/cart_pole.zip -d ~
```

Source **ROS 1 Noetic**:

```
31 source /opt/ros/noetic/setup.bash
```

Go to the *cart_pole* workspace directory and build the source folder:

```
32 cd ~/cart_pole/
33 catkin_make
```

Source the *cart_pole workspace*:

```
34 source ~/cart_pole/devel/setup.bash
```

Now run the *launch*-file and start the **Gazebo** simulation with the *Start*-button on the bottom control panel. Then start the *PID* controller:

```
35 roslaunch robot_launch launch_simulation.launch
36 roslaunch commander commander.launch
```

**RQT** is used to plot a desired topic, e.g. */cart_controller/state*:

```
37 rosrun rqt_plot rqt_plot
```

You can also change the parameters of the PID control:

```
38 gedit ~/cart_pole/src/commander/scripts/pid_commander.py
```

To see the effects of parameter changes, pause and reset the **Gazebo** simulation with *CTRL+R*, quit the *commander* and start all over.

A force can be applied to a *link* by choosing it in the *Models* tree. Just choose the *link*, *right-click* and apply the force. *rqt_reconfigure* can also be used to modify parameters.

**Note:** You might need to install the **ROS** packages *ros-noetic-effort-controllers* and *ros-noetic-ros-control* and the Python 3.7 module *deap*!

# D. Help & Suggestions

### Modifying render settings for Carla Simulator

There are a number of ways to tweak the performance of the *Carla Simulator*. The easiest way is to use one of the pre-settings *Low* or *Epic*.

```
39 /opt/carla-simulator/CarlaUE4.sh -quality-level=Low
```

Another way is to change its rendering parameters such as resolution, anti-aliasing etc. These can be found in

```
40 cd ~/.config/Epic/CarlaUE4/Saved/Config/LinuxNoEditor/
41 gedit GameUserSettings.ini
```

### Connecting python client remotely

If you happen to have two separate computers, you can use one to render the simulation and the other to compute the rest such as SLAM. For this, you must set

*ROS_MASTER_URI* and *ROS_IP* to your ROS master's IP, i.e.

```
42 export ROS_MASTER_URI=http://192.168.0.1
43 ROS_IP=192.168.0.1
```

as well as in the

```
44 gedit /etc/hosts
```

On the client side, set the ROS master's IP to *ROS_MASTER_URI* and *ROS_IP* to its own IP. Now change the host to the ROS master's IP in

```
45 manual_control.py
46 carla_ros_bridge_with_example_vehicle.py
```

### Modifying sensor settings

For the *Carla Simulator*, every parameter of the simulated sensors attached to the vehicle is defined in

```
47 gedit ~/carla_ws/src/ros-bridge/carla_spawn_objects/
   config/objects.json
```

For the SLAM algorithms **Gmapping** and **HectorSlam**, only the types, *rgb_view*, *lidar_raycast* and *lidar_raycast_semantics* are essential. Other sensors such as additional cameras, radar or GNSS can be deleted. Just save the original file as backup.

Set ranges must be equal throughout these sensors as well as in all the related *launch*-files:

```
48 ~/carla_ws/src/afrp2/launch
```

### Visualizing sensor data in real-time

**RVIZ** can be used to visualize sensor data stream:

```
49 rosrun rviz rviz
```

To visualize node and topic communication or transformations, *RQT* respective *tf2_tools* can be used:

```
50 rosrun rqt_tf_tree rqt_tf_tree
51 rosrun tf2_tools view_frames.py
```

### Changing Carla parameters during simulation

The **Carla** allows changes to the render engine on the fly. Try to change the map, weather or the number of pedestrians and cars:

```
52 python /opt/carla-simulator/PythonAPI/util/config.py --help
53 python /opt/carla-simulator/PythonAPI/util/config.py --map Town07
54 python /opt/carla-simulator/PythonAPI/util/config.py --weather
   WetCloudySunset
55 python /opt/carla-simulator/PythonAPI/util/environment.py --clouds
   100 --rain 100 --fog 50 --wetness 100 --puddles 60
56 python /opt/carla-simulator/PythonAPI/examples/spawn_objects.py -n
   50 -w 50 --safe
```