Assignment 1 Control for Self-Driving Cars

Due: 29.11.2023, 12:00AM

Introduction

The goal of this assignment is to get hands-on experience in implementing controllers for following a trajectory. Firstly, you will record a reference trajectory yourself by driving the car in the CARLA town using the keyboard and and then implement the following controllers to follow that trajectory:

1. PID based Longitudinal Controller
2. Geometrical Lateral Controller
3. Model Predictive Controller

Getting Started

1. If you use the computers in our lab, we did these steps for you and you can jump to step 4. If you use your own machine, install the CARLA server following the instructions provided in the first tutorial slides, which are available on eCampus.

2. Start the server and then the client to test your setup:

   For the server:

   ```
   $ cd /opt/carla-simulator/
   $ ./CarlaUE4.sh
   ```

   For the client:

   ```
   $ cd PythonAPI/examples
   $ python3 manual_control.py
   ```

3. Install our assets (needed to have customs traffic signs):

   ```
   $ cd /opt/carla-simulator/
   $ sudo wget http://www.ipb.uni-bonn.de/html/teaching/sdc/assets.tar.gz -P Import
   $ sudo ./ImportAssets.sh
   ```

4. Download `assignment1.zip` from eCampus, and unzip into current directory:

   ```
   $ unzip assignment1.zip -d .
   ```

5. **Only if you use your computer.**
   Make sure you have pip installed and upgraded:

   ```
   $ sudo apt install python3-pip
   $ pip3 install --upgrade pip
   ```

6. Finally, install the project with

```
$ cd assignment1
$ pip3 install -e .
```

Framework

For the assignments in this course, we provide you with a framework which manages the integration with the CARLA simulator and lets you focus on the implementation of a particular technique. The framework can be downloaded from eCampus as a `zip` file and has the following structure:

```
.
|-- ex1_control.pdf
|-- params.yaml
|-- pyproject.toml
|-- scripts
|   |-- evaluate_controller_performance.py
|   |-- main_control.py
|   |-- main_record_trajectory.py
|-- setup.py
|-- src
|   |-- pyilqr
|   |-- sdc_course
|       |-- control
|       |   |-- __init__.py
|       |   |-- controller.py
|       |   |-- decomposed.py
|       |   |-- pid.py
|       |   |-- mpc.py
|       |   |-- purepursuit.py
|       |   |-- stanley.py
|       |-- __init__.py
|       |-- perception
|       |-- planning
|       |   |-- __init__.py
|       |   |-- simple_global_planner.py
|       |   |-- simple_local_planner.py
|       |-- utils
|           |-- __init__.py
|           |-- car.py
|           |-- utility.py
|           |-- window.py
|           |-- world.py
|-- tests
    |-- test_control.py
```

Files you need to edit:

Files	Description
`pid.py`	Implementation of the PID controller.
`stanley.py`	Implementation of the Stanley controller.
`purepursuit.py`	Implementation of the Pure Pursuit controller.
`mpc.py`	Implementation of the MPC controller.
`params.yaml`	All the parameters related to your controller should be set here.

Files you might have a look:

Files	Description
`controller.py`	Abstract class of a controller.
`decomposed.py`	Special realization of a controller with separated lateral and longitudinal control.
`car.py`	Describes the car class and exposes the attributes you may need for completing your task
`world.py`	Sets up the CARLA world
`utility.py`	Consists of several helper functions which are used in the framework
`pyilqr`	Submodule that implements the iterative linear-quadratic regulator for nonlinear optimal control.

Control Tasks

As explained in the lecture, the longitudinal and lateral control of the vehicle can be done in a decomposed fashion with separate controllers or jointly by controlling the full state. In `controller.py`, you will find an `AbstractController` class that defines the basic layout of the controller. Both the `DecomposedController` and the `ModelPredictiveController` classes are inherited from this abstract class. For task 2 and 3, you will work with a decomposed controller implementing separate strategies for lateral and longitudinal control. In Task 4, you jointly compute the controls with a model predictive controller using the `pyilqr` library for optimal control.

Evaluation:

There are two ways to evaluate your homework. First, you can run our tests with

```
$ pytest tests/test_control.py --no-summary -s
```

Note that if you use a virtual environment like conda, virtualenv, etc., you need to install pytest inside the virutal environment as described here. Our tests will check for some dummy values if the "wiring" of your controllers is correct. You can also call the script directly with

```
$ python3 tests/test_control.py
```

which will fail as soon as one of the tests fails.

To test the performance of your (tuned) controllers, you can run `main_control.py` which automatically saves the tracked trajectory to `tracked_trajectory.txt` at the end. Evaluate the performance of your controller with the script `scripts/evaluate_controller_performance.py`.

```
$ python3 scripts/evaluate_controller_performance.py \
  -r results/recorded_trajectory.txt \
  -t results/tracked_trajectory.txt
```

This script will visualize the trajectory followed by the car using your controller against reference trajectory. It will also show your car's velocity against the reference velocity profile.

As a first step, visually inspect the two plots and check if your controller is able to follow the reference trajectory *closely*. This would mean checking:

1. Does your car have large oscillations around the reference trajectory?
2. How close is your is your car to the reference trajectory laterally (cross-track error)?
3. How close is your velocity to the reference velocity profile?

The evaluation script will also indicate whether you controller performance as **Good** or **Okay** or **Bad**. Make sure to tune the parameters such that you obtain either a **Good** or **Okay** performance. Try the evaluation script for the trajectory you recorded and make sure to obtain **Good** or **Okay** performance. We will later test your controller on our test trajectory.

Task 1: Record Reference Trajectory [25 Points]

In this task, you are asked to drive the car around manually using the keyboard in the CARLA town and record a trajectory. This trajectory should be used as the reference trajectory for further tasks. The outcome of this task is to generate a text file which consists the position (x, y) and the velocity (v) of the car.

Here how the output txt file should look like:

```
 50.664417,206.753799,13.721
-50.533421,206.782440,14.685
-50.394218,206.812851,15.591
```

Hints:

1. Figure out the keyboard controls by checking the manual control implementation in `sdc_course/utils/window.py`.
2. Ensure that the trajectory is long (and interesting) enough with several turns. The goal is to generate a trajectory that tests the performance of your controller.

Run the script:

```
$ python3 scripts/main_record_trajectory.py
```

to generate the reference trajectory. You can start recording the trajectory by pressing the `r` key and stop recording by pressing the `r` again.

Task 2: Longitudinal and Lateral PID Controller [25 Points]

The entry point for the remaining assignments is the `scripts/main_control.py` script. Execute the following command in order to start the simulation (in the folder you extracted the sources):

```
$ python3 scripts/main_control.py
```

When you run the `main_control.py` script, you should have the following visualization window:

In this window, you can toggle through different camera views by pressing the key `c`, which will also give you a bird's eye view. Implement two PID controllers (longitudinal and lateral) such that the car can follow the reference trajectory smoothly.

For this task, please only edit the `TODO` parts in the `pid.py` file. Most of the code to interface the controller with the CARLA simulation has been already written. Your task is to implement the main logic for the controller as indicated in the comments. The output of the longitudinal and lateral should be the throttle and steering commands respectively. These commands will then be executed by the car in the `scripts/main_control.py` if you select the option `pid` in the `params.yaml`.

Once you implement your controller, you can evaluate its performance with the script

```
$ python3 scripts/evaluate_controller_performance.py
```

Tune the parameters for the controller and set them in the `params.yaml` file.

Task 3: Geometric Lateral Controller [25 Points]

Implement a geometric lateral controller for the car. You may choose between the `Pure-Pursuit` controller or `Stanley` controller which have been explained in the lecture.

For this task, please only edit the corresponding methods in the `purepursuit.py` or `stanley.py` file. Similar to Task 2, your are asked to implement the main logic for the controller indicated in the comments.

To run your controller on the car, choose the appropriate control strategy in `params.yaml`. All the control parameters for your implementation should also be set there. In this task, we will use the PID controller developed in Task 2 for the longitudinal control.

Once you implement your lateral controller, you can evaluate its performance with the script `scripts/evaluate_controller_performance.py`. Make sure to tune the parameters such that you obtain either a **Good** or **Okay** performance.

Task 4: Model Predictive Controller (MPC) [25 Points]

Implement a model predictive controller for the car. For simplicity, we will use a unicycle model which assumes that a steering command directly influences the orientation. For this task, please only edit the `mpc.py` file. Implement the main logic for the controller indicated in the comments. You should use

the provided pyilqr library for defining the model dynamics, costs and the iLQR solver. The output should be a steering and acceleration command which is then executed by the car.

To run the MPC controller on the car, choose the control strategy as `mpc` in `params.yaml`. You can evaluate its performance with the script `scripts/evaluate_controller_performance.py`. Make sure to tune the parameters such that you obtain either a **Good** or **Okay** performance.

Hint: If you need some help how to use the pyilqr library, have a look at the examples in `pyilqr/tests`.

Submission:

Once you have completed your assignment by editing the relevant scripts in the framework, upload your solution as a `zip` file via eCampus with the filename `LastName1_LastName2.zip`. We will check the files that you need to edit and your recorded trajectory. Make sure that your solution is working, when we download a fresh framework and replace it with the abovementioned files.

Please make only one submission per team. You can also directly submit as a team in eCampus

Academic Dishonesty:

We will check your code against other submissions in the class. We will easily know if you copy someone else's code and submit it with some minor changes. We trust that you will submit your own work only. Please don't let us down. Note that in the case of plagiarism, you will be expelled from the course. We are strict on that.