If you are looking for information about the CPM Lab, consult the documentation.

Exercise 1. (*Setup*)

a) We recommend setting up Ubuntu on your computer. If you instead decide to use the provided virtual machine, you can skip this setup step.

Follow these instructions to clone and install the CPM Lab software for simulation. You will need MATLAB and Simulink with the DDS add-on, the Control System Toolbox, the Optimization Toolbox, and the System Identification Toolbox. If you need another toolbox later, they are easiest to install using the MATLAB installer.

b) Join your team's repository at GitLab to which you have been invited per email. You should not need to change any code outside of your repository. For smooth execution of your code in the lab, clone your repo such that you end up with the following folder structure (example for team 1, 't1'):

```
$HOME/dev
|-- software
    |-- cpm_lib
    |-- high_level_controller
    |-- ...
|-- lu2021
    |-- t1
        |-- ...
```

The path to your repository will be called `TEAMREPO` in the following. The path to the CPM Lab software will be called `CPMLAB`.

Initialize your repo with the code base given in sciebo (`lu_template`).

Exercise 2. (*Communication with and control of vehicles*)

a) How is the communication between components in the lab realized?

b) How can you read a vehicle's state in MATLAB?

c) The file `CPMLAB/cpm_lib/dds_idl/VehicleCommandDirect.idl` defines the message format of messages sent to control the vehicle in direct control mode. What are the contents of the interface definition language-file?

d) Execute the example `TEAMREPO/11_vehicle_basics/main_direct.m` to see the vehicle moving in a circle.

e) Repeat Task c) and Task d) for the path tracking mode.

f) Repeat Task c) and Task d) for the trajectory following mode.

g) Explain how the experimental concept for synchronization and determinism presented in the lecture is realized in the lab on the basis of the contents of a message.

Always ensure that the middleware's period in the lab control center "Parameters" tab is set according to your expectations for the control loop period in the high-level controller.

Exercise 3. (*Position control of path tracking vehicle*)
The goal of this exercise is that a vehicle follows a reference position on a path using path tracking mode. Use the folder `TEAMREPO/12_position_control` for this exercise.

a) Take a look at `TEAMREPO/0_common/get_path_points.m`. Understand how the path `'outer_lane'` is defined and use it from now on.

b) The vehicle needs to determine the distance traveled along the given path from the measured vehicle state. You can use the function `TEAMREPO/0_common/compute_rel_distance_on_path.m` for this.

c) Create a state-space model of a vehicle with the input v_{in}, and the outputs s and v, which are the distance traveled and the velocity, respectively. Feel free to use the System Identification Toolbox, e.g., with `ssest`. Provide the model in `TEAMREPO/0_common/get_longitudinal_model.m`. Save a plot with the input used for identification, the plant output and the model output. Draw a detailed block diagram of the control loop. Draw a sketch of the coordinate system(s) you are using in your model.

d) Follow the reference position s_{ref} given as

$$s_{\text{ref}}(t) = 1.1 \cdot t + 0.5 \cdot \sin t + s_0, \tag{1}$$

where $s_0 = s(t = 0)$ is the starting position of the controlled vehicle using model predictive control (MPC). Create an object of the class `ModelPredictiveControl` for your controller. The following constraints on the input should be considered

$$\begin{aligned} v_{\text{min}} &= 0\,\text{m/s}, & a_{\text{min}} &= -1\,\text{m/s}^2, \\ v_{\text{max}} &= 1.5\,\text{m/s}, & a_{\text{max}} &= 0.5\,\text{m/s}^2. \end{aligned} \tag{2}$$

Convert the constraint on the acceleration to a constraint on your input change and supply the MPC with a reference trajectory over the complete prediction horizon.