Simulation Exercise for 31390

Unmanned Autonomous Systems 31390
Department of Electrical Engineering • Technical University of Denmark
04 June 2020

PART 6: Gazebo Simulation

The following exercises will get you started on working with the Gazebo simulator together with Simulink.

All exercises will have the 3 fields: **Simulink template**, **Launch file**, and **Launch options**. These are to help you chose which of the simulink files and `roslaunch` parameters you should use to complete the exercise.

When using the simulator the default drone model will be the *Hummingbird* which is a '+' configuration drone. If you wish you can also use the 'x' configuration drone, *Ardrone*. If chosing to do so remember to rename the topics in the subscriber and publisher blocks from *hummingbird* to *ardrone*. Also consider the impacts of the different rotor configuration when modelling the drone. To use the *ardrone* in the remember to include:
`mav_name:=ardrone`
in the launch options.

Exercise 6.1: Handling waypoints in Simulink

The first exercise will make you familiar with the Gazebo / Simulink interface. Use the files in the `rotorS_position_interface` as a templete for the exercise. The Simulink model will provide you with a subscriber to get the current position and a subscriber to get the current attitude. Then you get a publisher so you can send desired positions to the drone simulator. You can also take a look at the `initial_test` simulink model to see how it is set up.

Using the position_interface model, make a model that can follow a sequence of points (route) stored in a vector. Make it so the next point is sent once the drone is currently within some distance, d, of the target point.

- Make a plot of the position together with the target positions using the following route vector with 4 waypoints:

```
route = [0 0 1; 3 0 2; -1 2 1; 1 -2 1]
```

where $route = [x_1, y_1, z_1; x_2, y_2, ...]$. Send the next target when the drone is within 10 cm of the target position.
Use yaw = 0°
Hint: You can have the full route as a constant and have a counter for handling the current point.

Simulink template: `rotorS_position_interface`
Launch file: `mav_with_position_controller.launch`
Launch options: None

Exercise 6.2: Making a hovering attitude controlled drone

In this exercise we will make a stable hovering controller for your attitude controlled drone. For this we will be using the attitude controlled drone. When using the attitude controlled drone we have 4 control inputs - roll, pitch, yawrate and thrust. For this exercise we are only going to make and tune the thrust controllers.

- Make a height controller so you can control the height width a z reference in meters.
- Make a step response of the height with your controller for a step from 0 til 1 meter.

  When running the simulation set the input of roll/pitch/yawrate to 0.

  Thrust is an value ranging from -15 to 15 with 0 being approximately hovering thrust, 15 being full thrust and -15 being motors off.

Simulink template: `rotorS_attitude_interface`
Launch file: `mav_rpy_thrust_controller.launch`
Launch options: None

Exercise 6.3: Attitude controlled position controller

Using the work form Exercise 6.2 upgrade the controller to include a position controller using the attitude commands. Remember to also have a yaw controller.

- Make a plot of a 2 meter step response in x
- Make a plot of a 2 meter step response in y
- Make a plot of a 90° step response of the yaw controller$x$

  Perform all steps while hovering at 1 meter

Simulink template: `rotorS_attitude_interface`
Launch file: `mav_rpy_thrust_controller.launch`
Launch options: None

Exercise 6.4: Testing the position controller with a simple route

Expand your model from Exercise 6.3 so that it can follow a sequence of points. You could combine it with your work from Exercise 6.1.

- Make a plot of your position controller where you follow the following route and the corresponding yaw position(in degrees):
- The plot should show the current position and attitude together with the current setpoint value.

```
route = [0 0 1; 2 0 1; 2 2 1; 0 2 1; 0 0 0]
yaw = [0; 90; 180; -90; 0]
```

Simulink template: `rotorS_attitude_interface`
Launch file: `mav_rpy_thrust_controller.launch`
Launch options: None

Exercise 6.5: Making a hovering drone from motor velocities

Now we are going to have the full drone control done in Simulink. This means that we are going to be sending the desired motor velocities to the Gazebo Simulator.
For this exercise use the `rotorS_motor_velocities_interface` as a template. See the Simulation instruction document for motor numbering in Gazebo.

- Make a hovering controller (height controller)
- Make a plot of a step response from 0 to 1 meter.

Simulink template: `rotorS_motor_velocity_interface`
Launch file: `mav_without_controller.launch`
Launch options: None

Exercise 6.6: Closing the attitude control loop

With a hovering controller working from Exercise 6.5 extend your controller to include attitude control.

- Make a plot of a roll step response of 10°
- Make a plot of a pitch step response of 10°
- Make a plot of a yaw step response of 90° Make the plots while hovering at 1 meter.

Simulink template: `rotorS_motor_velocity_interface`
Launch file: `mav_without_controller.launch`
Launch options: None

Exercise 6.7: Closing the position control loop on top of the attitude controller

With a attitude controller working from Exercise 6.5 repeat Exercises 6.3 and 6.4 using your attitude controller instead of the simulators.

Simulink template: `rotorS_motor_velocity_interface`
Launch file: `mav_without_controller.launch`
Launch options: None

Exercise 6.8: 2D Path following

In this exercise we will combine the work from Exercise 6.1 with path planning. Run your A* on the 2D map for `maze_1`. From your work with path planning use your path planning function to find a path in `maze_1`, see Figure 1.

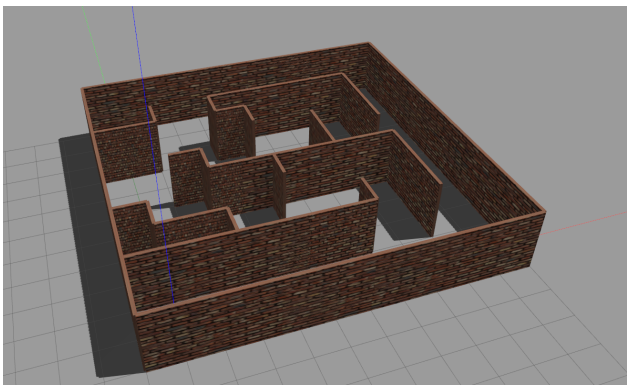

Figure 1: `Maze_1`

`Maze_1` is a 12x12x2 meter maze with an inner grid if 10x10x1 so that it can be used for 2D path planning and following. Running the `maze_1.m` file produces a variable, `map`, that contains the information about walls in the maze. A 1 indicates a wall and a 0 a free path. `map(1,1)` will give you the information about point $(0, 0)$ in the maze because of MatLab using 1-indexing - remember this when converting your route through the map into position commands for the drone. The map indexing is `map(x,y)`.

- Make the drone go from $(0, 0)$ in world coordinates to $(3, 6)$ using your A* search and route following scheme. Fly the route at a height of 1 meter.
- Make a xy-plot of the flown path with the planned route.

Simulink template: `rotorS_position_interface`
Launch file: `mav_with_position_controller.launch`
Launch options: `world_name:=maze_1`

Exercise 6.9: 3D Path following

This exercise is similar to the previous but this time we will move to 3D path planning and following. You will be working with the `maze_3D`, see Figure 2. `Maze_3D` is a 12x12x6 tower with an inner grid of 10x10x3. The usable flying heights in the maze is 1, 3, and 5 meters corresponding to the 3 levels in the map. So `map(2,2,2)` would correspond to the point $(1, 1, 3)$. Remember to account for this shift when passing the route to drone.

Use your 3D A* algorithm to fly around in the maze.

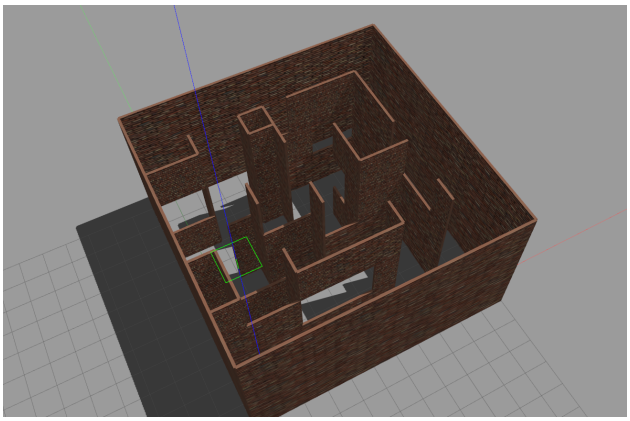

Figure 2: `Maze_3D`

- Make the drone go from $(0, 0, 1)$ in world coordinates to $(8, 9, 5)$ using your A* search and route following scheme.
- Make a 3D-plot of the flown path with the planned route.

Simulink template: `rotorS_position_interface`
Launch file: `mav_with_position_controller.launch`
Launch options: `world_name:=maze_3D`

Exercise 6.10: Combining your work

The last exercise is to test the the controllers you developed in Exercise 6.3+6.4 or Exercise 6.5-6.7 in the maze.

If you have made a working Simulink position controller(either from 6.3+6.4 or 6.5-6.7) repeat Exercise 6.9 using that controller.

Compare with the results you got from exercise 6.9.