Tampere University

Fundamental of Mobile Robot
AUT-710

Exercise 5

Muhamad Fikri

Widhi Atman

15.3.2024, **SA205,** 10:15 - Finish

Tampere University

General Plan for Exercises

- Exercise 4: Implementation of model + Basic Control
- Exercise 5: Collision Avoidance with SI model
  - Point obstacle – switching behavior with obstacle avoidance
  - non-point obstacle – switching behavior with wall-following
  - Point obstacles – QP-based controller

10 point

20 point

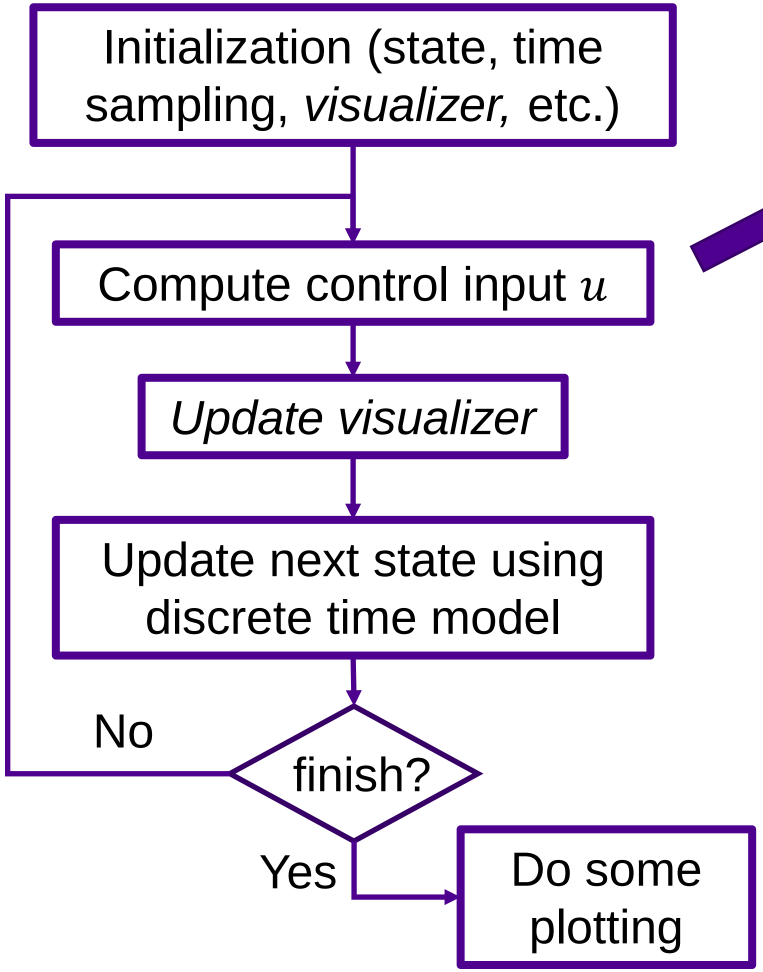
Deadline: Monday 01.04.2024 at 23:59

- Exercise 6: Control of Unicycle
- Mini Projects (optional)

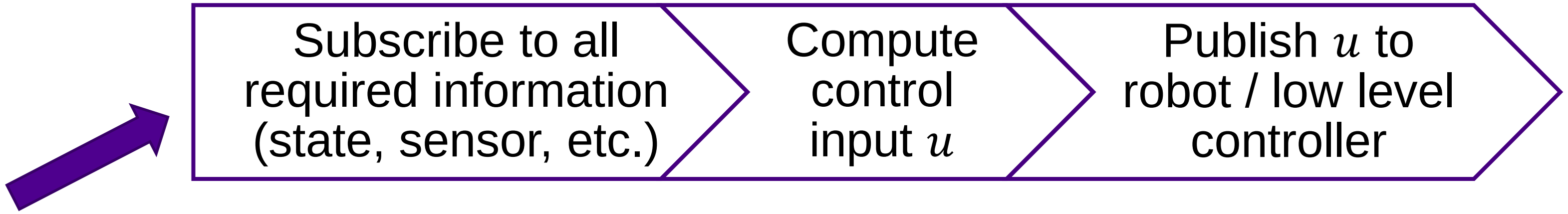
20 point

Bonus up to 20 point

Flowchart of Simulator



If you are interested in implementing this to ROS

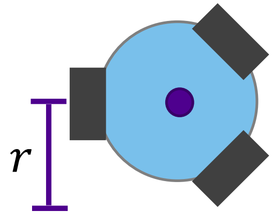


Specifications (for Exercise 5)

Time sampling $T = 10ms$

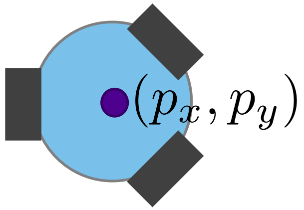
Robot's radius = $0.21\ m$

Max translational vel. $\left(v_x^2 + v_y^2\right)^{\frac{1}{2}}$ = 0.5 m/s
Max rotational vel. ($|\omega|$) = 5 rad/s

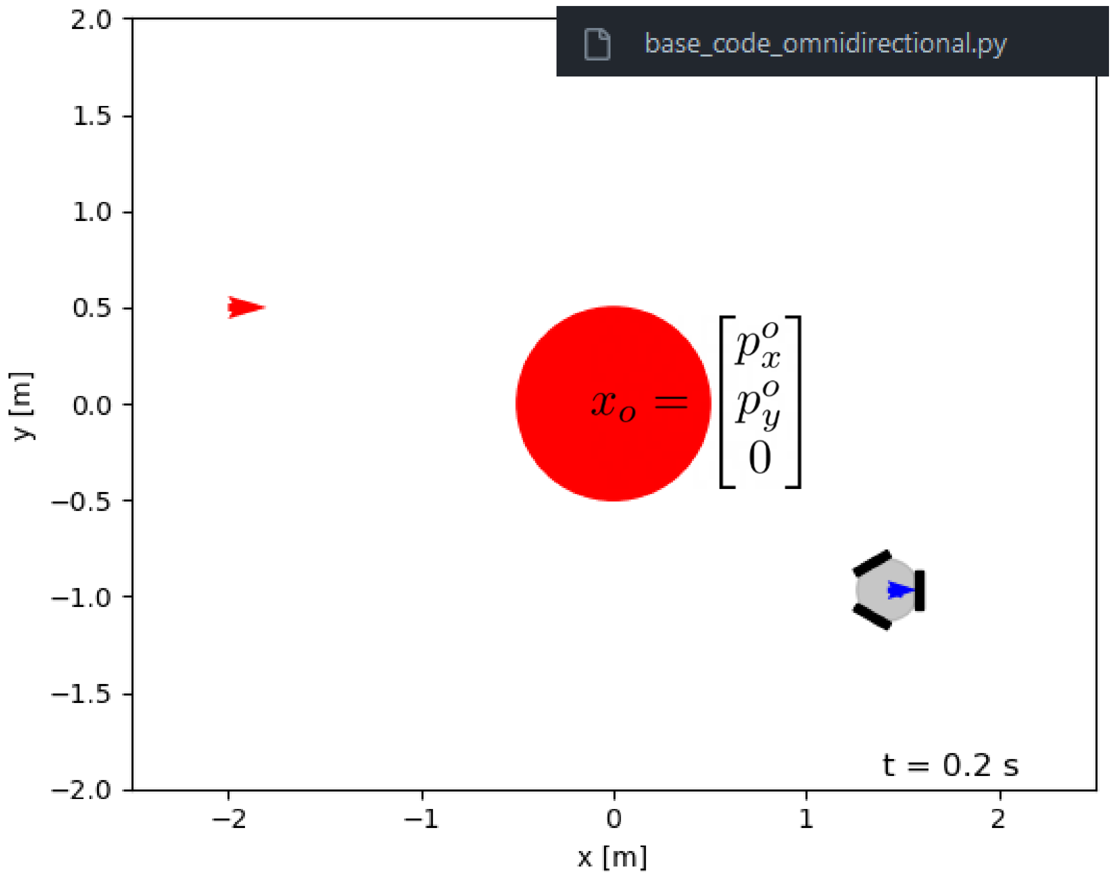


Tampere University

Exercise 5.1 – Scenario



$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix}$ $u = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$



Model: omnidirectional mobile robot (**single-integrator model**)
Initial Position: $x[0] = [1.5 \quad -1 \quad *]^T$
Goal: static at $x^d = [-2 \quad 0.5 \quad 0]^T$.
* Can be any orientation at goal position

Key Scenario:

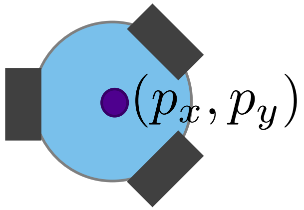
- We assume the robot/controller can identify a **circular obstacle** (centroid and radius) once it is near. Here, the obstacle is centered at $p_x^o = 0$, $p_y^o = 0$ with radius $0.5m$.

Control Objective:

- Reach the goal while avoiding contact/collision with obstacle

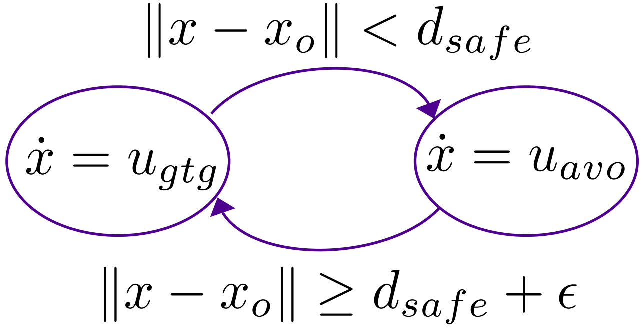
Exercise 5.1 – Task

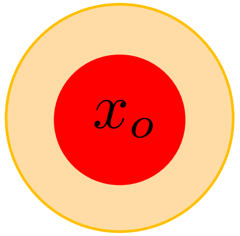
5 points



$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix} \quad u = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

By taking account of the **robot's size and limitation**, design and implement the following **switched controller**





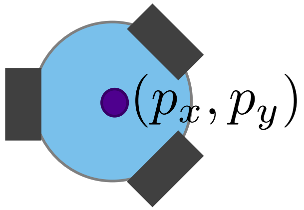
Describe your approach in designing u_{gtg}, u_{avo}, d_{safe}, and ϵ, as well as your observation on the resulting controller.

Show the result by plotting:

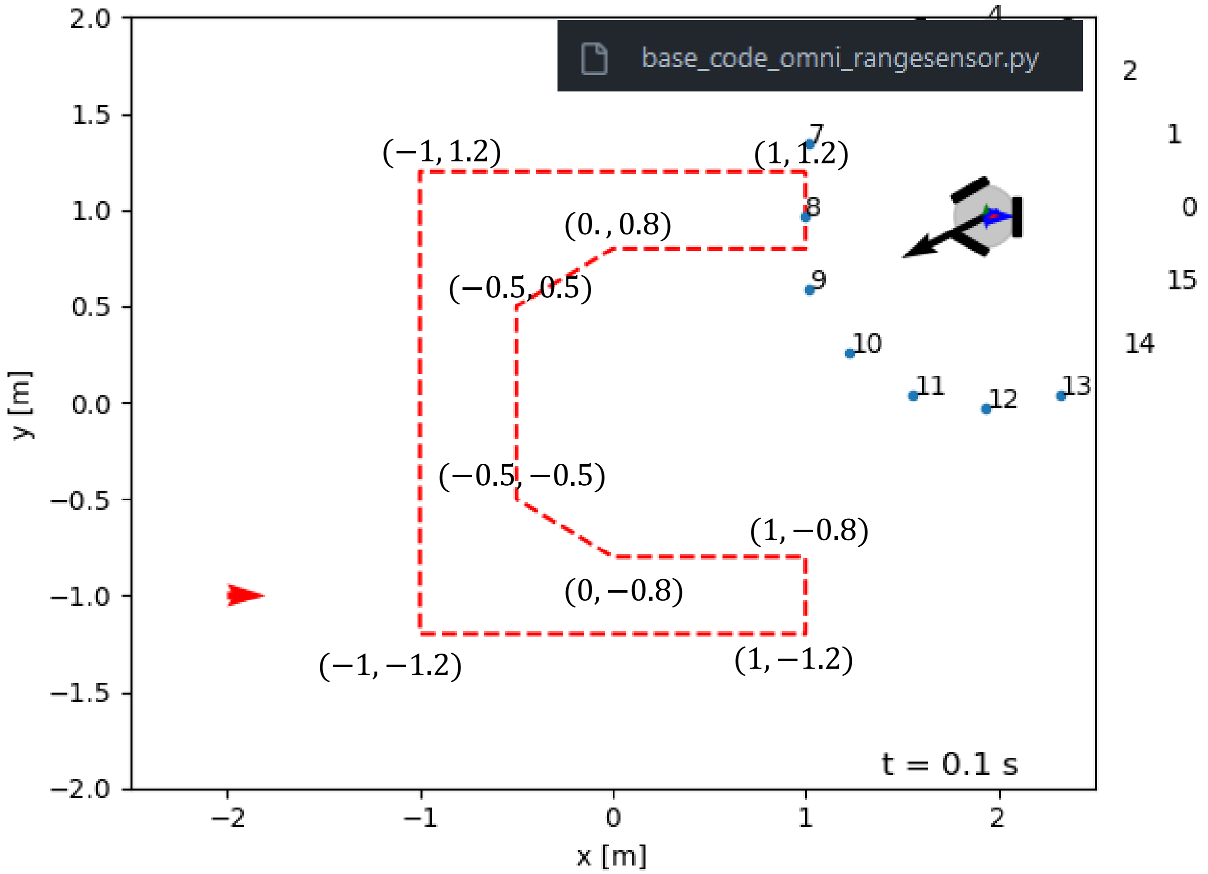
- *time series* of control input u and $\left(v_x^2 + v_y^2\right)^{0.5}$
- *time series* of error $\left(x^d - x\right)$,
- *time series* of distance to obstacle $\|x - x_o\|$
- *time series* of state trajectory x vs x^d, and
- XY **trajectory** of the robot (or final snapshot of the simulator).

Deadlock issue: *Try to set the initial position to* $x[0] = [2 \quad -0.5 \quad 0]^T$ *and see what happen.*

Exercise 5.2 – Scenario



$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix} \quad u = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$



Model: omnidirectional mobile robot (**single-integrator model**)

Initial Position: $x[0] = [2 \quad 1 \quad 0]^T$

Goal: static at $x^d = [-2 \quad -1 \quad *]^T$.

* Can be any orientation at goal position

Key Scenario:

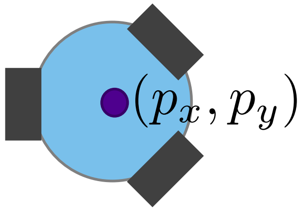
- An obstacle presents in the field, but ***unknown*** to the robot/controller.
- The obstacle is detected by the reading from range sensor ($< 1\ m$)

  Assume accurate readings from sensors.

Control Objective:

- Reach the goal while avoiding contact/collision with obstacle

Tampere University



$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix} \quad u = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

Exercise 5.2 – Task

10 points

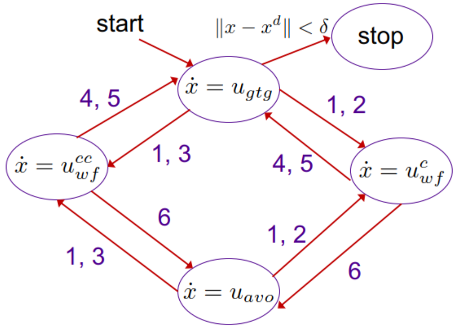
By taking account of the **robot's size and limitation**, design and implement **wall-following behavior** to your switching controller in 5.1.

Describe your approach in designing u_{wf}^{c}, u_{wf}^{cc}, and computing x_o (and u_{avo}) from sensor readings as well as your observations on the resulting controller.

Show the result by plotting:

- *time series* of control input u and $\left(v_x^2 + v_y^2\right)^{0.5}$
- *time series* of error $\left(x^d - x\right)$,
- *time series* of *minimum reading distance* from the sensor
- *time series* of state trajectory x vs x^d, and
- XY **trajectory** of the robot (or final snapshot of the simulator).

Putting all together



Conditions for switching:

(1) $d_{safe} - \epsilon \le \|x - x_o\| \le d_{safe} + \epsilon$

(2) $u_{gtg}^T u_{wf}^c > 0$

(3) $u_{gtg}^T u_{wf}^{cc} > 0$

(4) $u_{avo}^T u_{gtg} > 0$

(5) $\|x(t) - x^d\| < \|x(t_s) - x^d\|$

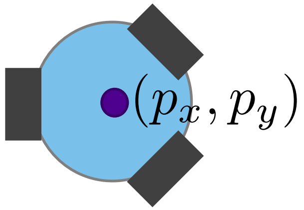
(6) $\|x - x_o\| < d_{safe} - \epsilon$

IMPORTANT TIPS:

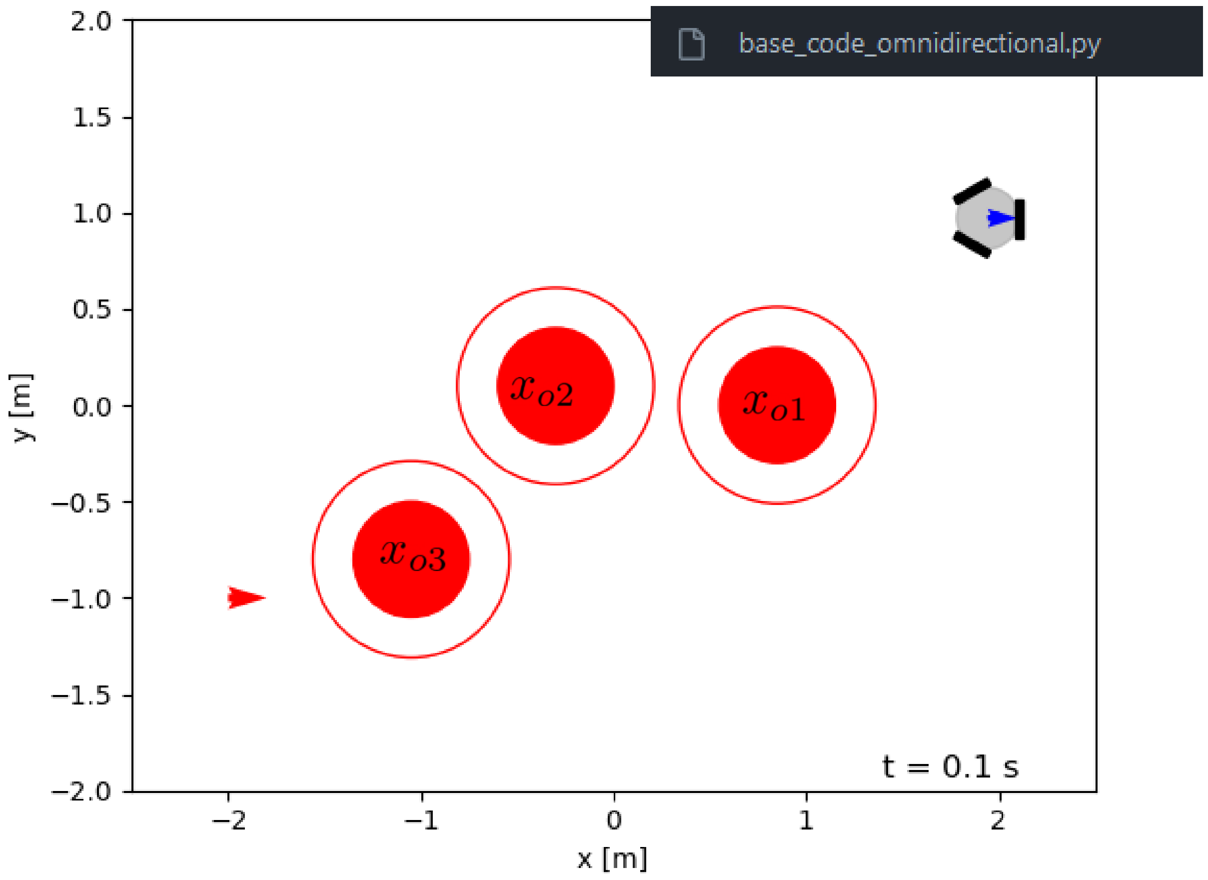
- Use visualization to help debug (e.g., the u_{gtg}, u_{avo}, u_{wf}^{c}, u_{wf}^{cc}, etc.)
- *Print* every time the state changes

Tampere University

Exercise 5.3 – Scenario



$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix} \quad u = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$



Model: omnidirectional mobile robot (**single-integrator model**)
Initial Position: $x[0] = [2 \quad 1 \quad 0]^T$
Goal: static at $x^d = [-2 \quad -1 \quad *]^T$.
* Can be any orientation at goal position

Key Scenario:

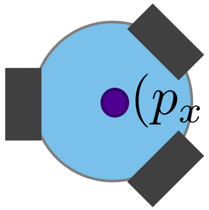
- We assume the robot/controller can identify a **circular obstacle** (centroid and radius) once it is near. Here, 3 obstacle presents:
  1. centered at $p_x^{o1} = 0.85$, $p_y^{o1} = 0$ with radius $0.3m$.
  2. centered at $p_x^{o2} = -0.3$, $p_y^{o2} = 0.1$ with radius $0.3m$.
  3. centered at $p_x^{o3} = -1.05$, $p_y^{o3} = -0.8$ with radius $0.3m$.

Control Objective:

- Reach the goal while avoiding contact/collision with obstacle

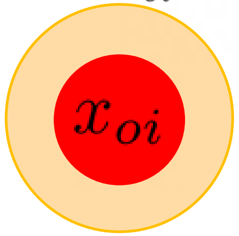
Exercise 5.3

5 point



$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix} \quad u = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

$$x_{oi} = \begin{bmatrix} p_x^{oi} \\ p_y^{oi} \\ 0 \end{bmatrix}$$



By taking account of the robot's limitation, implement the **QP-based controller** as follows

$$
\begin{aligned}
u = \arg\min_{u^*} \; & \|u_{gtg} - u^*\|^2 \\
\text{s.t.} \quad & \left(\frac{\partial h_{o1}}{\partial x}\right)^T u^* \geq -\gamma(h_{o1}(x)) \\
& \left(\frac{\partial h_{o2}}{\partial x}\right)^T u^* \geq -\gamma(h_{o2}(x)) \\
& \left(\frac{\partial h_{o3}}{\partial x}\right)^T u^* \geq -\gamma(h_{o3}(x))
\end{aligned}
$$

with $h_{oi} = \left\| \begin{bmatrix} p_x \\ p_y \end{bmatrix} - \begin{bmatrix} p_x^{oi} \\ p_y^{oi} \end{bmatrix} \right\|^2 - R_{si}^2$

Use $R_{si} = 0.51$ for all obstacles.

NOTE: u_{gtg} still need to comply with the robot's max speed. You can use u_{gtg} from 5.1.

Then, test the controller for $\gamma(h) = 0.2h$, $\gamma(h) = 10h$, and $\gamma(h) = 10h^3$ and describe your observation on what does the variation affects.

Show the result by comparing the plot for each γ function variation via:

- Time series comparison of control input u_{gtg} vs u (plot only v_x and v_y)
- Time series comparison of function h_{o1}, h_{o2}, and h_{o3}
- Comparison of XY **trajectory** of the robot

How to show obstacle on Simulator?

Plot using the axis object in the sim_mobile_robot class → sim_visualizer.ax

Option 1: using patch in matplotlib (for simple shape: circle, rectangle, etc.)

```
if IS_SHOWING_2DVISUALIZATION: # Initialize Plot
    sim_visualizer = sim_mobile_robot( 'omnidirectional' ) # Omnidirectional Icon
    #sim_visualizer = sim_mobile_robot( 'unicycle' ) # Unicycle Icon
    sim_visualizer.set_field( field_x, field_y ) # set plot area
    sim_visualizer.show_goal(desired_state)
    sim_visualizer.ax.add_patch( plt.Circle( (0, 0), 0.5, color='r' ) )
    sim_visualizer.ax.add_patch( plt.Circle( (0, 0), d_safe, color='r', fill=False) )
    sim_visualizer.ax.add_patch( plt.Circle( (0, 0), d_safe + eps, color='g', fill=False) )
```

Example for 5.1

Draw (full) red circle for obstacle
Draw empty red circle for d_{safe}
Draw empty green circle for $d_{safe} + \epsilon$

Option 2: by defining obstacle's vertices and use line plot

```
if IS_SHOWING_2DVISUALIZATION: # Initialize Plot
    sim_visualizer = sim_mobile_robot( 'omnidirectional' ) # Omnidirectional Icon
    #sim_visualizer = sim_mobile_robot( 'unicycle' ) # Unicycle Icon
    sim_visualizer.set_field( field_x, field_y ) # set plot area
    sim_visualizer.show_goal(desired_state)

    # Display the obstacle
    obst_vertices = np.array( [ [-1., -1.2], [1., -1.2], [1., 1.2], [-1., 1.2], \
            [-1., 0.8], [0., 0.8], [0.5, 0.5], [0.5, -0.5], [0., -0.8], [-1., -0.8], [-1., -1.2] ])
    sim_visualizer.ax.plot( obst_vertices[:,0], obst_vertices[:,1], '--r' )
```

Example for 5.2

Tampere University

How to implement QP for 5.3?

$$u = \arg\min_{u^*} \|u_{gtg} - u^*\|^2$$
$$\text{s.t.} \quad \left(\frac{\partial h_{o1}}{\partial x}\right)^T u^* \geq -\gamma(h_{o1}(x))$$
$$\left(\frac{\partial h_{o2}}{\partial x}\right)^T u^* \geq -\gamma(h_{o2}(x))$$
$$\left(\frac{\partial h_{o3}}{\partial x}\right)^T u^* \geq -\gamma(h_{o3}(x))$$

Refer to lecture 10 on 22.3.2023

$$\min_z \frac{1}{2} z^T Q z + c^T z$$
$$\text{s.t.} \quad Hz \preceq b$$

```
import cvxopt
```

Require ***cvxopt*** python package

```
# QP-based controller
# ---------------------------------------------------------------------
Q_mat = 2 * cvxopt.matrix( np.eye(2), tc='d')
c_mat = -2 * cvxopt.matrix( u_gtg[:2], tc='d')

# Fill H and b based on the specification afterwards
# --> row is number of constraints / specification
H = np.zeros([row, 2]) # TODO
b = np.zeros([row, 1]) # TODO

# Resize the H and b into appropriate matrix for optimization
H_mat = cvxopt.matrix( H, tc='d')
b_mat = cvxopt.matrix( b, tc='d')

# Solving Optimization
cvxopt.solvers.options['show_progress'] = False
sol = cvxopt.solvers.qp(Q_mat, c_mat, H_mat, b_mat, verbose=False)

current_input = np.array([sol['x'][0], sol['x'][1], 0])
```

$$z = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

Change these two lines to the appropriate values

NOTE: These code only compute the linear velocity (x and y) which is sufficient for 5.3.

Question?

- Consult them via
  - Exercise sessions on 15.3.2024, 22.3.2024, and 29.3.2024
  - Teams channel