Fundamental of Mobile Robot AUT-710

Exercise 5

Widhi Atman

18.3.2022, 10:15 - Finish

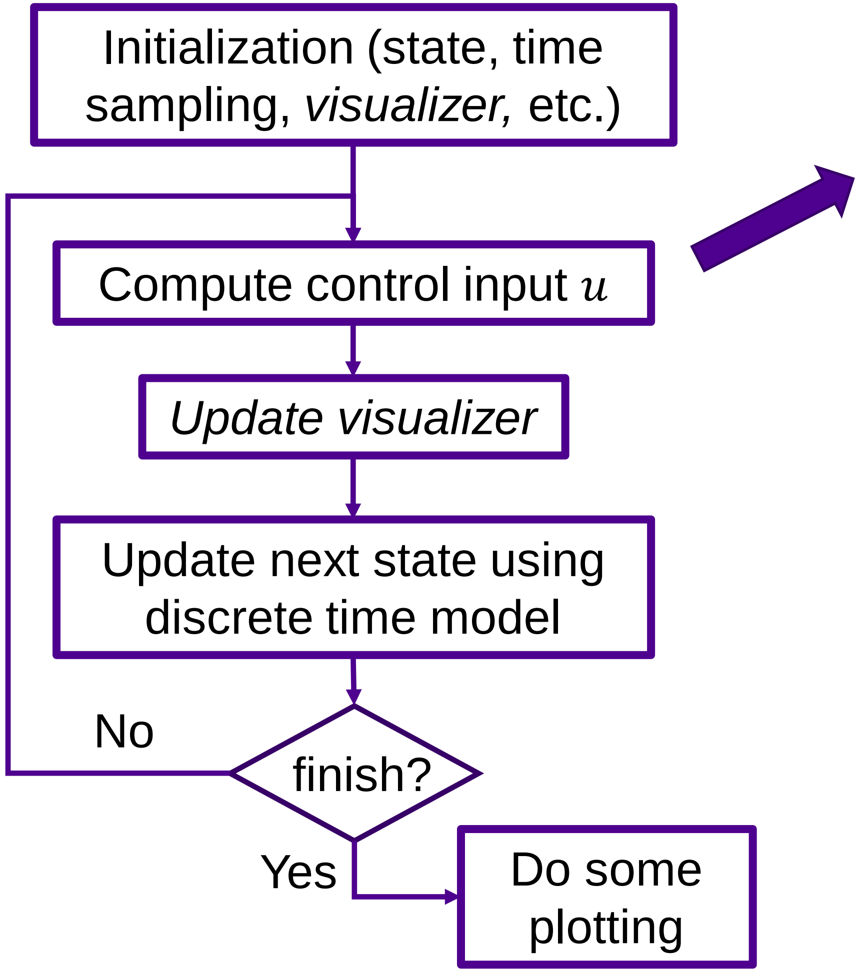
General Plan for Exercises

- Exercise 4: Implementation of model + Basic Control — 10 point
- Exercise 5: Collision Avoidance with SI model — 20 point
  - Point obstacle – switching behavior with obstacle avoidance
  - non-point obstacle – switching behavior with wall-following
  - Point obstacles – QP-based controller

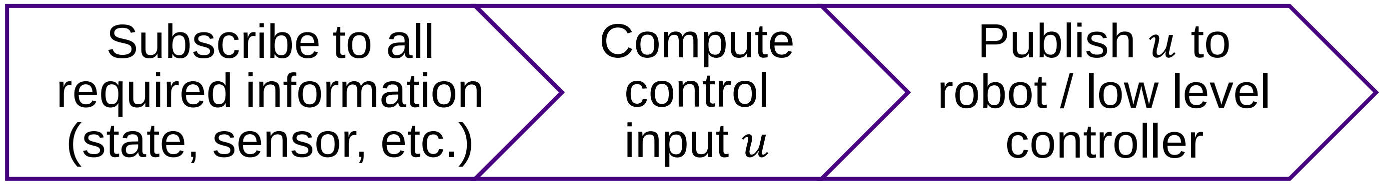
Deadline: Friday 1.4.2022 at 10:00

- Exercise 6: Control of Unicycle — 20 point

Flowchart of Simulator



If you are interested in implementing this to ROS



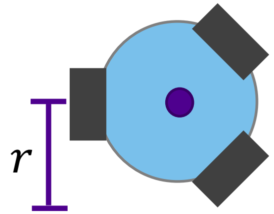
Specifications (for Exercise 5)

Time sampling $T = 10ms$

Robot's radius = $0.21\ m$

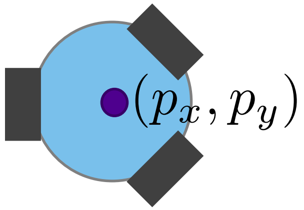
Max translational vel. $\left(v_x^2 + v_y^2\right)^{\frac{1}{2}}$ = 0.5 m/s

Max rotational vel. ($|\omega|$) = 5 rad/s

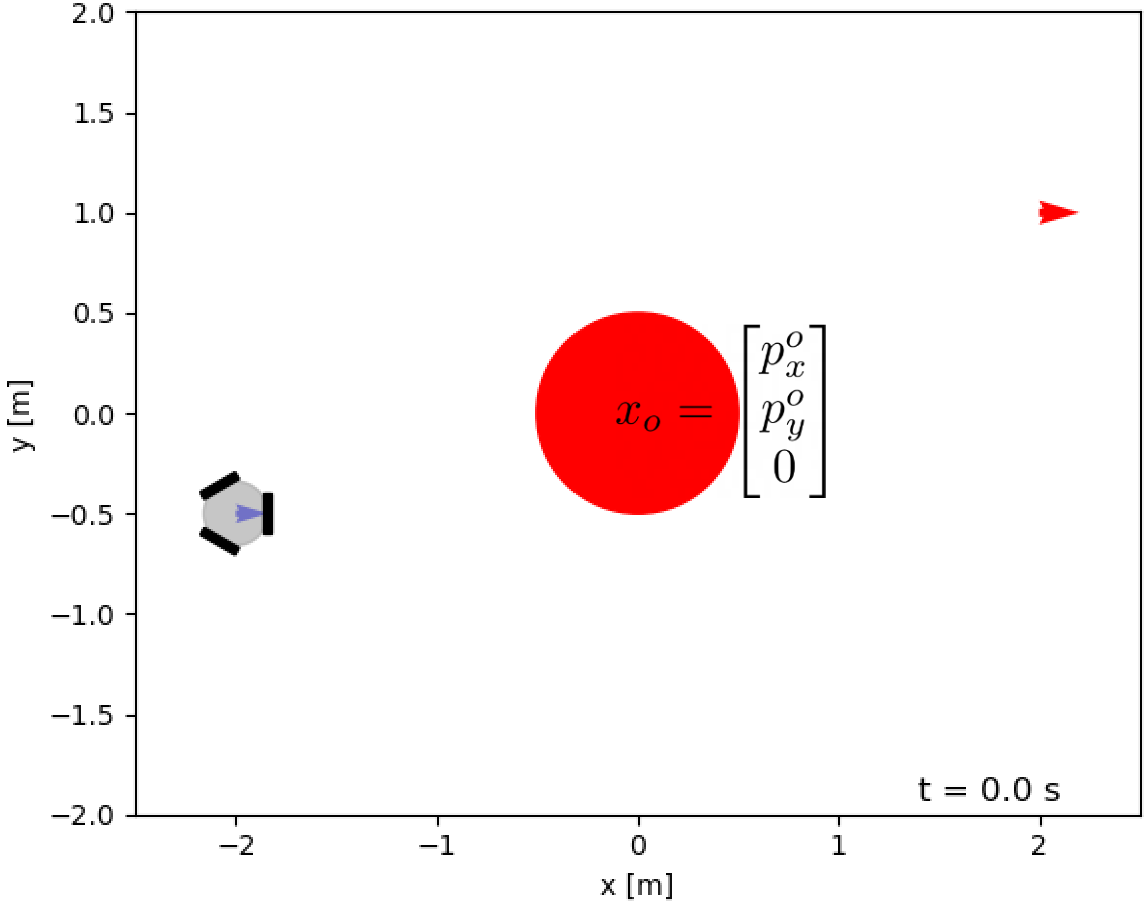


Tampere University

Exercise 5.1 – Scenario



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



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

Key Scenario:

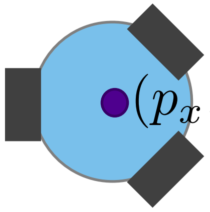
- We assume the robot/controller can identify a **circular obstacle** (centroid and radius) once it is near. Here, the obstacle is centered at $p_x^o = 0$, $p_y^o = 0$ with radius $1m$.

Control Objective:

- Reach the goal while avoiding contact/collision with obstacle

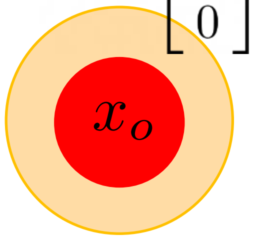
Exercise 5.1 – Task

5 points

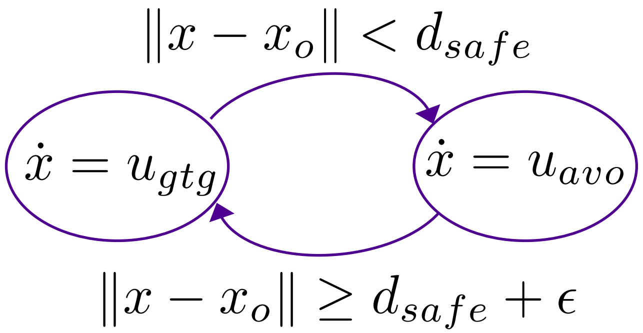


$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix} \quad u = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

$$x_o = \begin{bmatrix} p_x^o \\ p_y^o \\ 0 \end{bmatrix}$$



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.

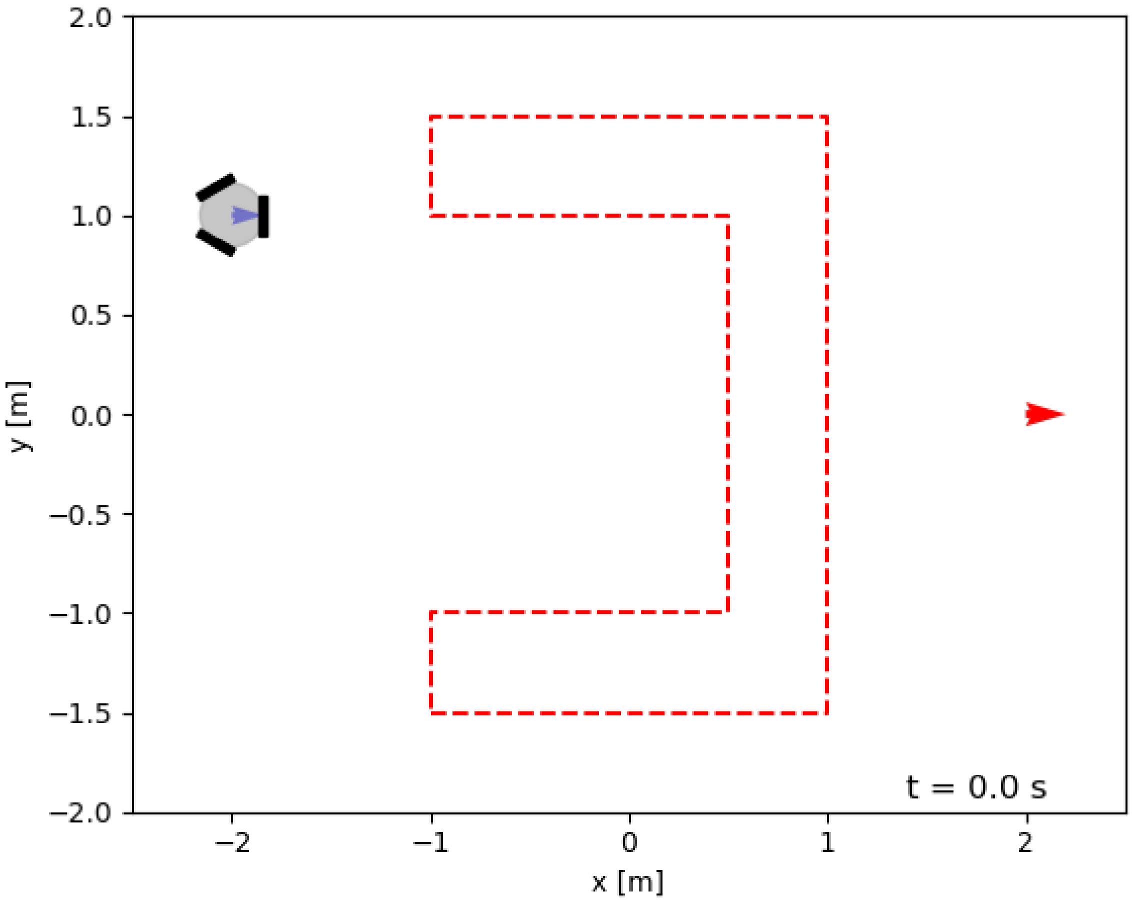
Show the result by plotting:

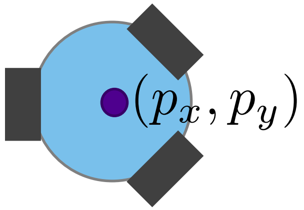
- *time series* of control input u and $\left(v_x^2 + v_y^2\right)^{0.5}$
- *time series* of error $\left(x^d - x\right)$,
- *time series* of distance to obstacle $\|x - x_o\|$
- *time series* of state trajectory x vs x^d, and
- XY **trajectory** of the robot (or final snapshot of the simulator).

Deadlock issue: *Try to set the initial position to* $x[0] = [-2 \quad -1 \quad 0]^T$ *and see what happen.*

Tampere University

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 $x^d = [2 \quad 0 \quad *]^T$.
* Can be any orientation at goal position

Key Scenario:

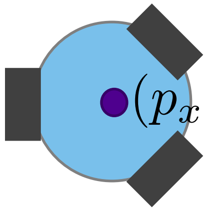
- An obstacle presents in the field, but ***unknown*** to the robot/controller.
- The obstacle is detected by the reading from range sensor ($< 1\,m$)
  Assume accurate readings from sensors, simulated by detect_obstacle.py (described in later slide).

Control Objective:

- Reach the goal while avoiding contact/collision with obstacle

Exercise 5.2 – Task

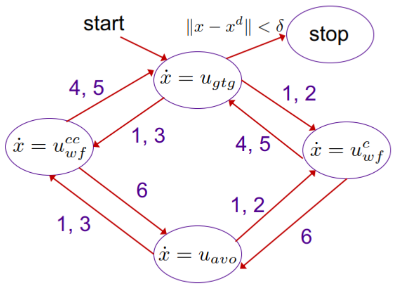
10 points



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

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

Putting all together



Conditions for switching:

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

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

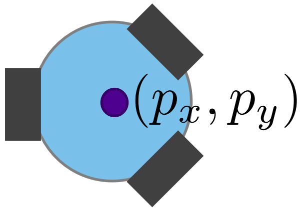
Describe your approach in designing u_{wf}^c, u_{wf}^{cc}, and computing x_o from sensor readings as well as your observations on the resulting controller.

Show the result by plotting:

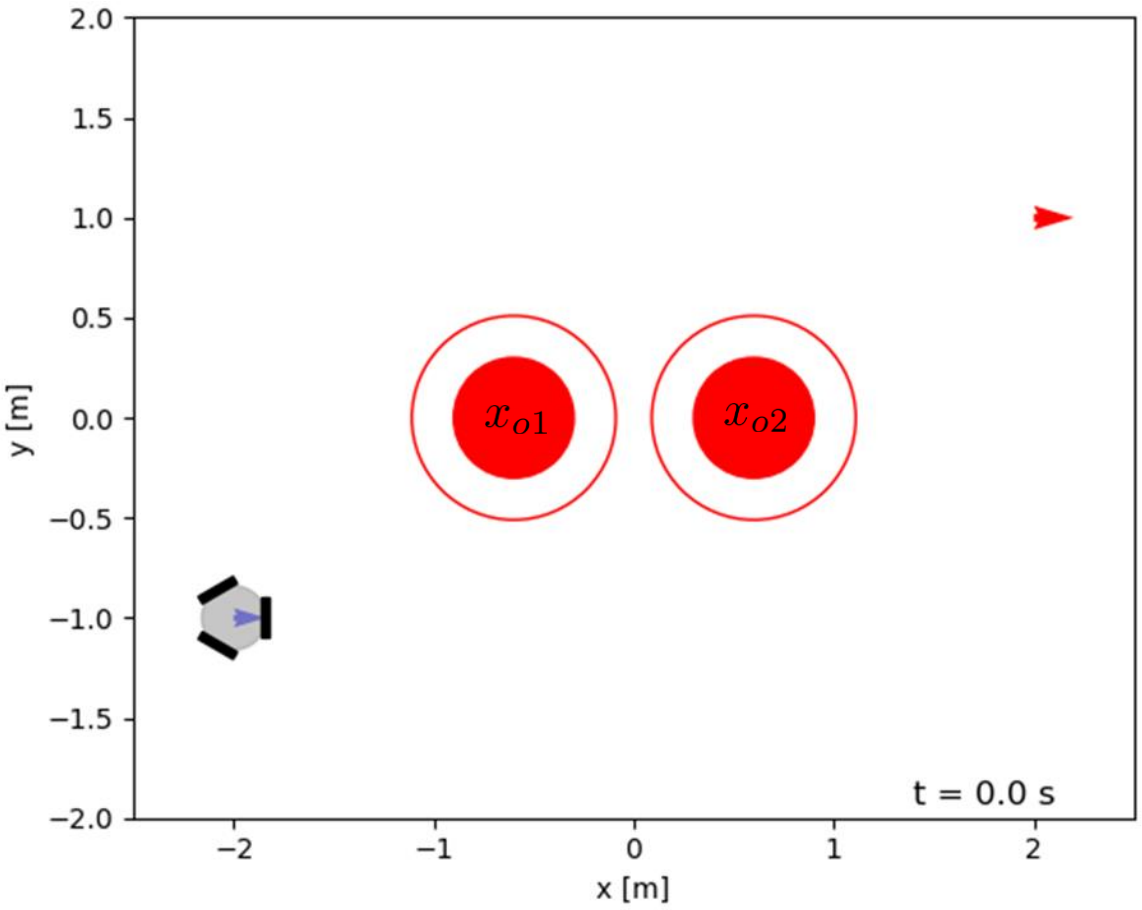
- *time series* of control input u and $\left(v_x^2 + v_y^2\right)^{0.5}$
- *time series* of error $\left(x^d - x\right)$,
- *time series* of state trajectory x vs x^d, and
- XY **trajectory** of the robot (or final snapshot of the simulator).

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

Exercise 5.3 – 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: $x[0] = [-2 \quad -1 \quad 0]^T$
Goal: static at $x^d = [2 \quad 1 \quad *]^T$.
* Can be any orientation at goal position

Key Scenario:

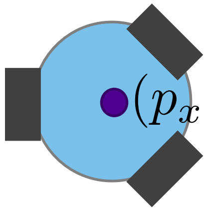
- We assume the robot/controller can identify a **circular obstacle** (centroid and radius) once it is near. Here, two obstacle presents:
  1. centered at $p_x^{o1} = -0.6$, $p_y^{o1} = 0$ with radius $0.3m$.
  2. centered at $p_x^{o2} = 0.6$, $p_y^{o2} = 0$ with radius $0.3m$.

Control Objective:

- Reach the goal while avoiding contact/collision with obstacle

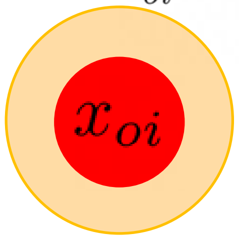
Exercise 5.3

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

$$u = \underset{u^*}{\arg\min} \, \|u_{gtg} - u^*\|^2$$

$$\text{s.t.} \quad \frac{\partial h_{o1}}{\partial x} u^* \leq \gamma(h_{o1}(x))$$

$$\frac{\partial h_{o2}}{\partial x} u^* \leq \gamma(h_{o2}(x))$$

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 both 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} and h_{o2}
- 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 $d_{safe} + \epsilon$

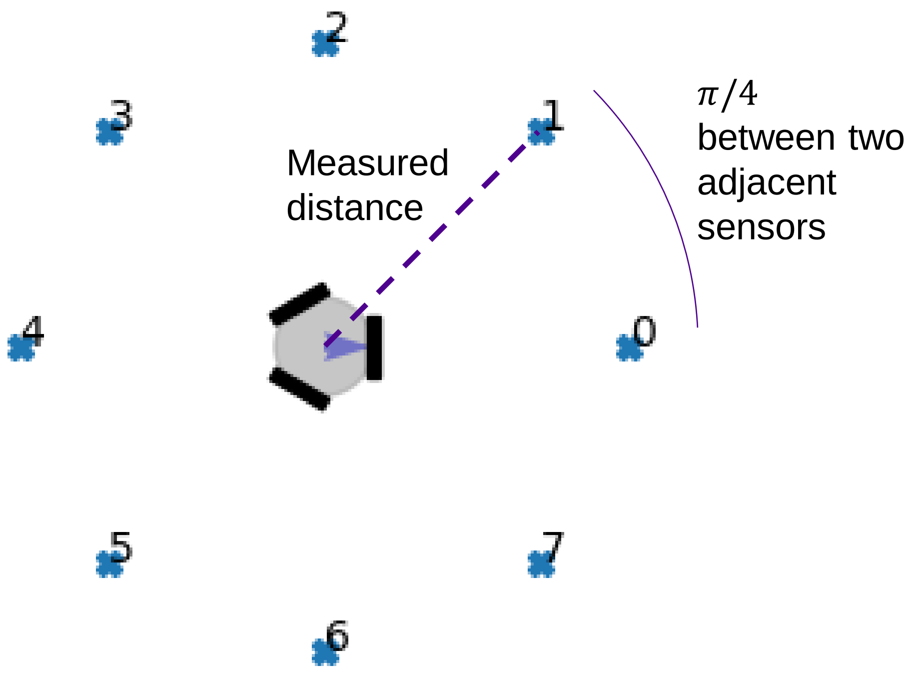
Option 2: by defining obstacle's vertices and use line plot

```
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)

    # ordered position of vertices in [x, y]
    obst_vertices = np.array( [ [-1., -1.5], [1., -1.5], [1., 1.5], [-1., 1.5], \
            [-1., 1.], [0.5, 1.], [0.5, -1.], [-1., -1.], [-1., -1.5] ])
    sim_visualizer.ax.plot( obst_vertices[:,0], obst_vertices[:,1], '--r' )
```

Example for 5.2

How to detect obstacle for 5.2?



NOTE: You don't need to detect obstacle for 5.1 and 5.3. For 5.1 and 5.3, it is sufficient to use the same base code as in exercise 4.

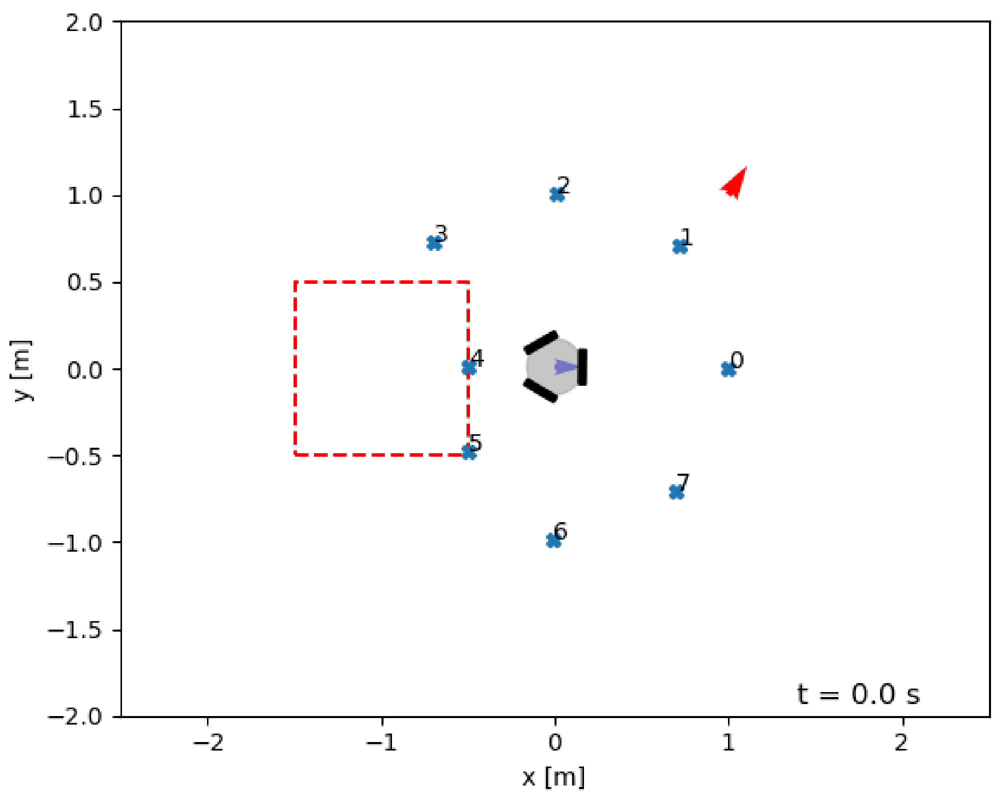
```
from detect_obstacle import detect_obstacle_e5t2
```

```
# Get information from sensors
sensors_dist = detect_obstacle_e5t2( robot_state[0], robot_state[1], robot_state[2])
```

Requirement: *Shapely* python package
Input: p_x, p_y, θ (in this specific order)
Output: array with 8 distance values of the sensor reading ***measured from the robot's center.***
The returned value will be between collision range to max sensing range (no obstacle detected)

Important points:
- Max sensing range: 1m
- Minimal distance to object (collision range): 0.21m
- **The code will crash** if the robot is inside the obstacle or within collision range.

How to detect obstacle for 5.2? (cont'd)



NOTE: You don't need to detect obstacle for 5.1 and 5.3. For 5.1 and 5.3, it is sufficient to use the same base code as in exercise 4.

What to do with the sensor reading?

- If the **value** is the Max sensing range (i.e., 1)
  → no obstacle is detected

Reduced to avoid rounding issue

- If the **value** is less than Max sensing range (i.e., < 0.99)
  → obstacle is detected.
  To compute the detected point in the obstacle

$$\begin{bmatrix} x_o^W \\ 1 \end{bmatrix} = \hat{R}(\underbrace{x^T}_{\text{robot's position in \{W\} frame}},\theta)\overbrace{\hat{R}((\underbrace{x_s^B}_{\text{sensor's position in \{B\} frame}})^T,\underbrace{\theta_s^B}_{\text{sensor's orientation in \{B\} frame}})\begin{bmatrix} x_o^S \\ 1 \end{bmatrix}}^{x_o^B}$$

Remark:

$$\underbrace{\hat{R}(p^T,\theta) = \begin{bmatrix} \overset{\text{rotation}}{R(\theta)} & \overset{\text{translation}}{p} \\ 0 & 1 \end{bmatrix}}_{\text{3x3 matrix}} \qquad R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\boxed{x_o^S = \begin{bmatrix} value \\ 0 \end{bmatrix} \quad x_s^B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}}$$

- **If the code is crashed** → Read the error, most probably the robot is too close to obstacle (simulate collision with obstacle)

How to implement QP for 5.3?

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

Refer to lecture 23.3.2022

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

```
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])
```

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.

Any Question?

Small note: Try to think of what you want mobile robot to do.
I am preparing a mini group project on Exercise 6 (or 7)
where you are required to utilize/combine any tools from this course
in a free-to-decide scenario (given a certain guidelines).

*Or what you have learned but
haven't tried in the exercise yet.