Fundamentals of Mobile Robots

Exercise Report Number: 5

by:

Dong Le (151057054)

Date: 10.04.2023

THIS PAGE LEFT BLANK INTENTIONALLY

Dong Le

List of Terminology

u_gtg	Control input go-to-goal
u_avo	Control input avoidance
u_wf_cc	Control input wall-following counter clockwise
u_wf_c	Control input wall-following clockwise
eps	Epsilon
d_safe	Area around obstacle
x_o_1 and x_o_2	Two sensors are chosen to calculate u_wf
v_wf_c and v_wf_cc	Two vectors of wall-following based on u_avo, the purpose to check condition only

Dong Le

Table of Contents

Dong Le

Introduction

This exercise is based on the concepts proportional and hard switching and quadratic programming theory.

Proportional control is a control system technology based on a response in proportion to the difference between what is set as a desired process variable (or set point) and the current value of the variable.

Hard switching is a method that changing the controller based on conditions and controller's state.

The purpose of this exercise is to learn how to design and implement controller using different methods to get the robot to reach the desired goal position and avoiding obstacles.

In the first problem we implement proportional control and avoidance for hard switching method.
In the second problem we use sensor reading to switch between methods.
In the third problem we design an optimized control input based on the quadratic programming theory.

Dong Le

Methodology

Problem 1

The theory of proportional control was used to design control input to reach the goal and the theory of control input avoidance to avoid obstacle on the way. The theory of omnidirectional mobile robot.

First, the initial robot state and desired state are defined. Then using 3 zero matrices to store data from the process to plot after all. The current control input was calculated based on the distance from the robot and the obstacle, so an appropriate switch controller was applied. A robot state at that time was made by adding time step times control input into the previous robot state. Note to design d_safe and eps to avoid collision and infinity switches in a finite time, and saturation the speed to satisfy the robot's limitation.

Some libraries are used in the python code such as Matplotlib, numpy, a custom library called "visualize_mobile_robot" and a pre-made python code called "base_code_omnidirectional.py"

Problem 2

The theory of wall following was used to track the moving goal and designing proportional control. The theory of omnidirectional mobile robot.

The initial robot state and desired state are defined. Before compute the control input for the system, defined a control state as "gtg" and a radius $\|x(t_s) - x^d\|$ as 0. To compute the control input, divided into 4 parts: preparing, checking condition, checking control state, and computing control input base on control state.

Firstly preparing, defined six conditions as false, find the minimum reading distance and its sensor coordinate. Found out the distance from robot to the minimum sensor, and from robot to the goal. Computed the u_gtg and u_avo based on those data.

Secondly, found out the v_wf_c and v_wf_cc based on u_avo and 2 rotaion matrices, then checking 6 conditions.

Thirdly, checking the conditions and control state in order to change the control state, and whenever the control state changed to "wf_c" or "wf_cc", a new value of radius $\|x(t_s) - x^d\|$ was set to compute the second stage.

Finally, based on the control state from the previous step, compute control input. If the control state is "wf_c" or "wf_cc" at first it needs to compute two sensors, then calculate control input u.

A robot state at that time was made by adding time step times control input into the previous robot state. The speed of the robot was needed to saturate to satisfy the robot's limitation.

Some libraries are used in the python code such as Matplotlib, numpy, a custom library called "visualize_mobile_robot" and a pre-made python code called "base_code_omnidirectional.py"

Dong Le

Problem 3

The theory of proportional control for orientation was used to reach the desired goal. The theory of Quadratic Programming to find an optimized control input. The theory of omnidirectional mobile robot.

The initial robot state and desired state are defined. The current control input was calculated based on the Quadratic Programming and a control input go-to-goal. Firstly, computed a control input u_gtg, then applying a QP-based controller with 3 obstacles. Changing $\gamma(h)$ to check the robot's behaviour. A robot state at that time was made by adding time step times control input into the previous robot state. Note to saturation the speed to satisfy the robot's limitation.

Some libraries are used in the python code such as Matplotlib, numpy, a custom library called "visualize_mobile_robot" and a pre-made python code called "base_code_unicycle.py"

Dong Le

Results and Discussion

Problem 1

In this problem, omnidirectional mobile robot was used with the position (p_x, p_y)

$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix} \tag{1}$$

And the control input:

$$u = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{2}$$

Initial position $x[0] = [-2 \;\; 0.5 \;\; 0]^T$

The goal $x^d = [2 \;\; -1 \;\; 0]^T$

There is a circular obstacle which is centered at $p_x^0 = 0$, $p_y^0 = 0$ with radius 0.5m.

The purpose of the task is to reach the goal while avoiding collision with obstacle

In this task, the control input was definded as two control inputs go-to-goal and avoidance

$$u_{gtg} = k_g(x^d - x), k_g > 0 \tag{3}$$

$$u_{avo} = k_o(x - x_o), k_o > 0 \tag{4}$$

Considering the robot's size and limitation, a safe zone was created around the obstacle. The distance from the boundary of the safe zone to the edge of the obstacle must be greater than haft of the robot's size.

A value epsilon was created also to avoid the robot while switching between 2 control inputs goes too far from the goal, and to prevent infinitely many switches in finite times.

$$d\_safe = 0.75$$

$$eps = 0.01$$

For u_gtg control input, the proportional control k_g was designed based on the equation

$$k_g = \frac{v_0\left(1 - e^{-\beta\|\bar{e}\|}\right)}{\|\bar{e}\|} \tag{5}$$

With $\|\bar{e}\|$ is the magnitude of the error between the desired state and the current state

$$\|\bar{e}\| = \theta_d - \theta \tag{6}$$

v_0 was set to 5, and β was set to 0.5. The value of k_g was depended on the value of v_0 and β.

For u_avo control input, the proportional control k_o was designed based on the equation

$$k_o = \frac{1}{\|\bar{e}\|}\left(\frac{c}{\|\bar{e}\|^2 - eps}\right), c > 0 \quad (7)$$

With $\|\bar{e}\|$ is the magnitude of the error between the current state and the obstacle state

$$\|\bar{e}\| = \theta - \theta_o \quad (8)$$

c was set to 5

The figure 1 shows the condition to switch between 2 control inputs was dependent on the d_safe and epsilon

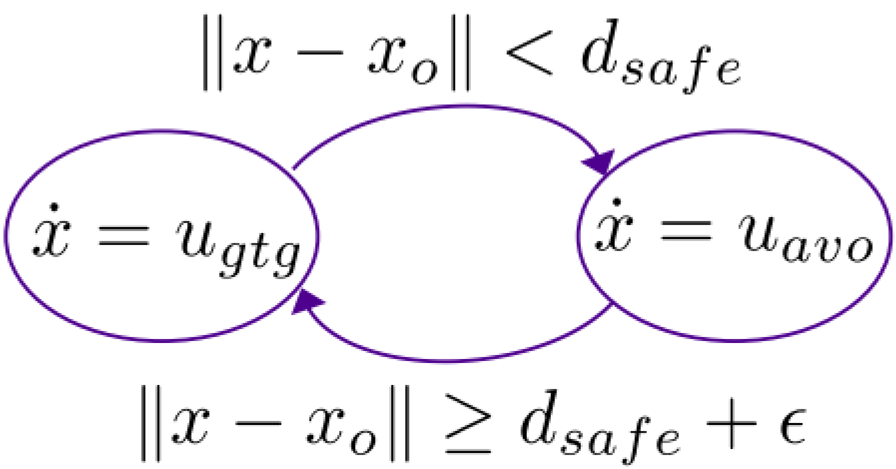

Figure 1: Condition to switch between two control inputs

Due to the fact that maximum translation velocity of the robot $\sqrt{(v_x^2 + v_y^2)} = 0.5\ m/s$

Therefore, after one iteration, the speed v_x and v_y must reduce to satisfy the limitation of the robot.

$$v_x = \frac{v_x * 0.5}{\sqrt{(v_x^2 + v_y^2)}} \quad (9)$$

$$v_y = \frac{v_y * 0.5}{\sqrt{(v_x^2 + v_y^2)}} \quad (10)$$

Figure 2 shows the trajectory of the robot after applying two control inputs.

Dong Le

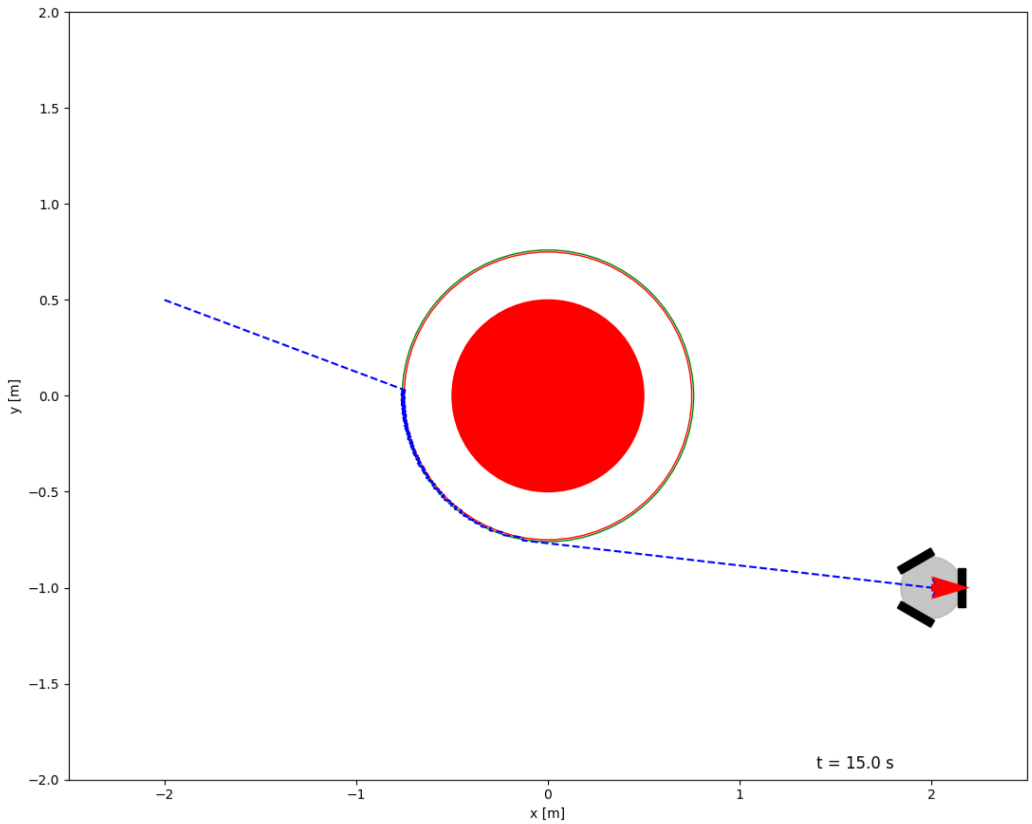

Figure 2: Robot trajectory

Figure 3 shows the control input against time and the speed of the robot $\sqrt{(v_x^2 + v_y^2)}$.

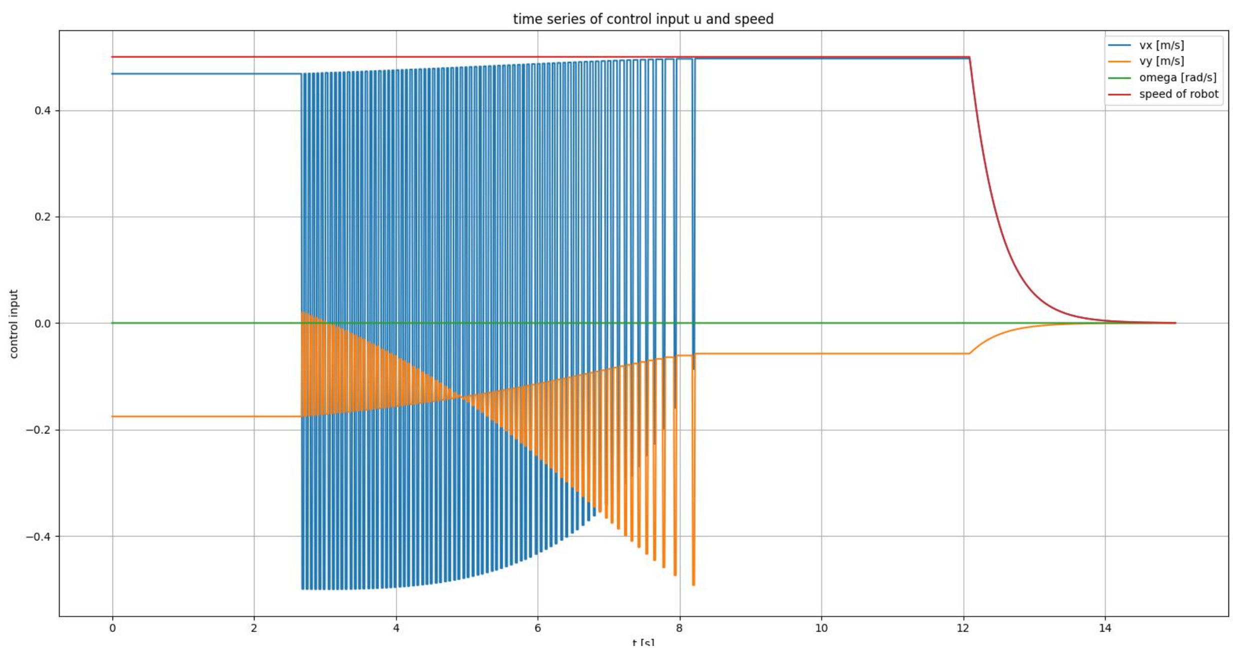

Figure 3: Control input and speed

By applying the formula 9 and 10, the robot's translation velocity is always less or equal than 0.5
Figure 4 shows the error between the goal state and the robot state

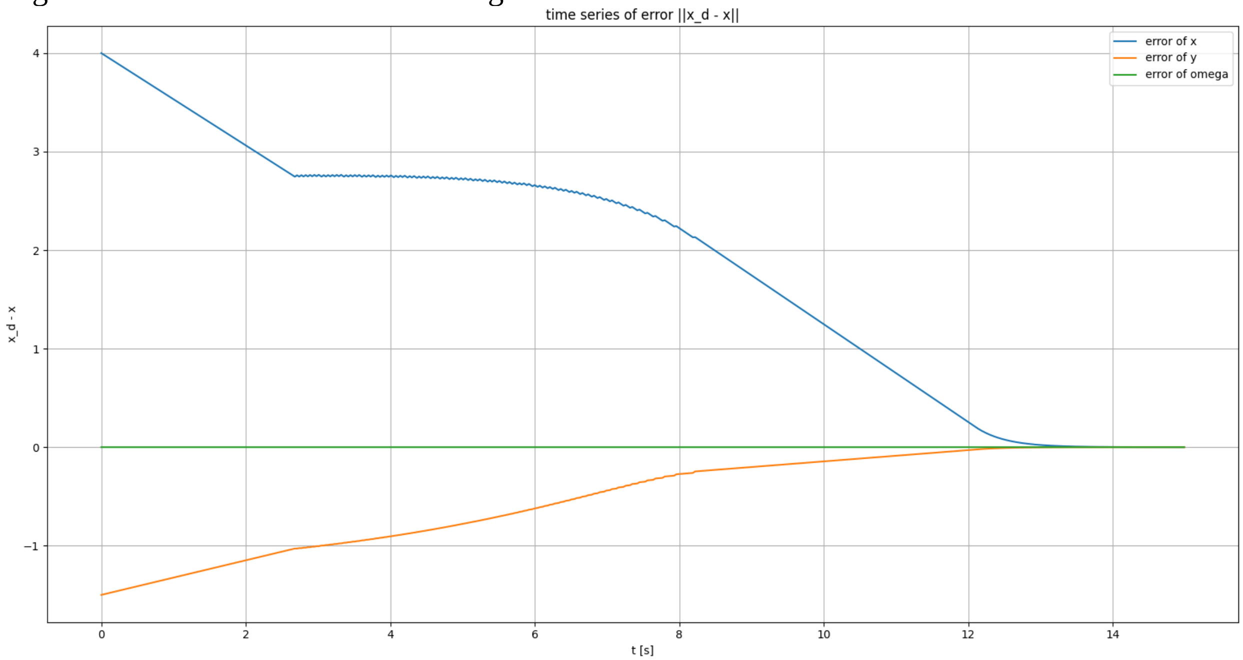

Figure 4: time series of error ||x_d - x||

After nearly 13 seconds, the robot reached the goal, so all error between the goal state and the robot state went to 0.

Figure 5 shows the distance between the robot state and the obstacle

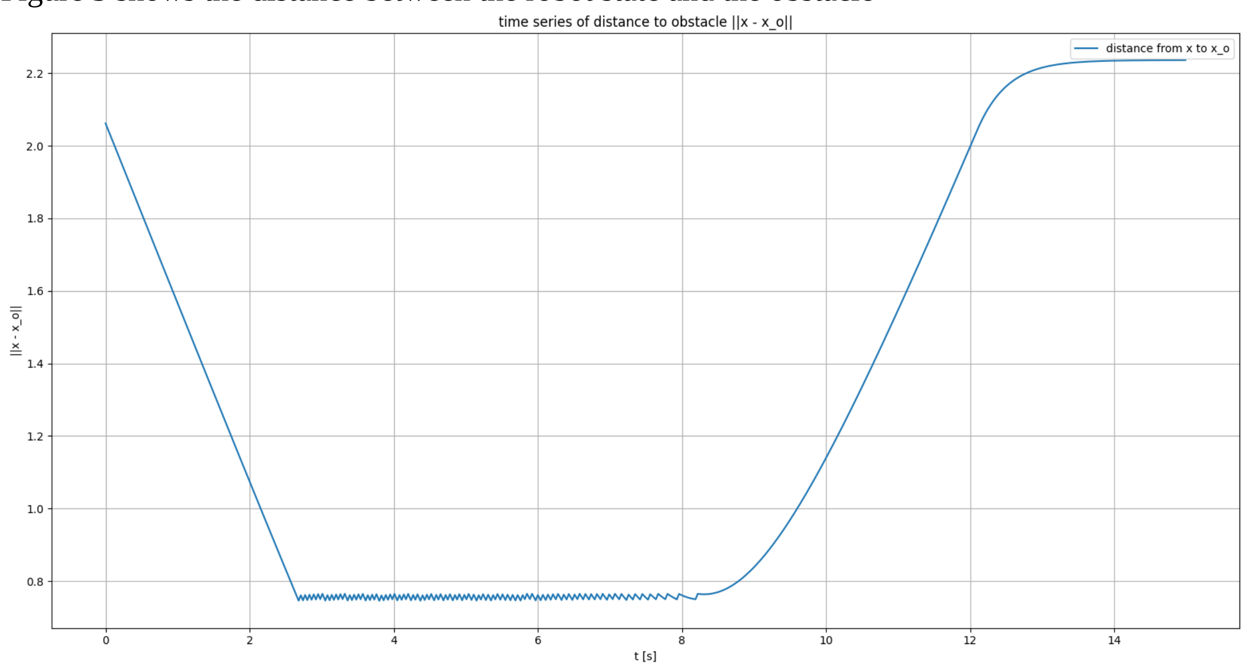

Figure 5: time series of distance to obstacle

The time duration between 2.5 to 8.5 was the time the robot switching between two control inputs, thanks to the d_safe and eps, the distance between robot and the obstacle was not 0, which avoid collision.

Figure 6 shows the state trajectory of the robot state and the goal state, which is similar to the figure 4.

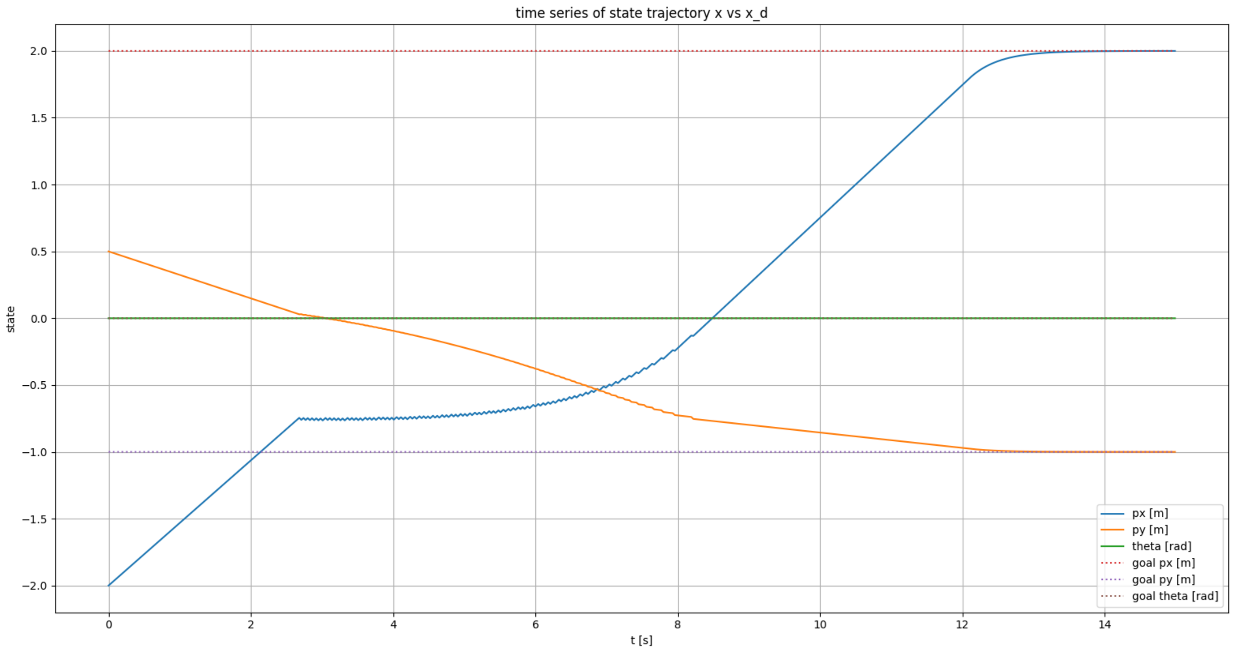

Figure 6: time series of state trajectory x and x_d

In conclusion, the switched controller was easy to apply, it does not need expensive calculations. Even though the time to reach the goal is high, it eventually reaches the final position. Moreover, d_safe and epsilon were significantly important in this approach to avoid collision between the robot and the obstacle. However, figure 3 displays the control input of the controller, it was changing constantly during the time the robot approached the d_safe boundary, it was not a smooth ride.

Addition to the origonal problem, a deadlock issue was mentioned, a new initial position was set $x[0] = [-2\ \ 1\ \ 0]^T$

Figure 7 shows the robot trajectory and it was stuck when approaching the obstacle

Dong Le

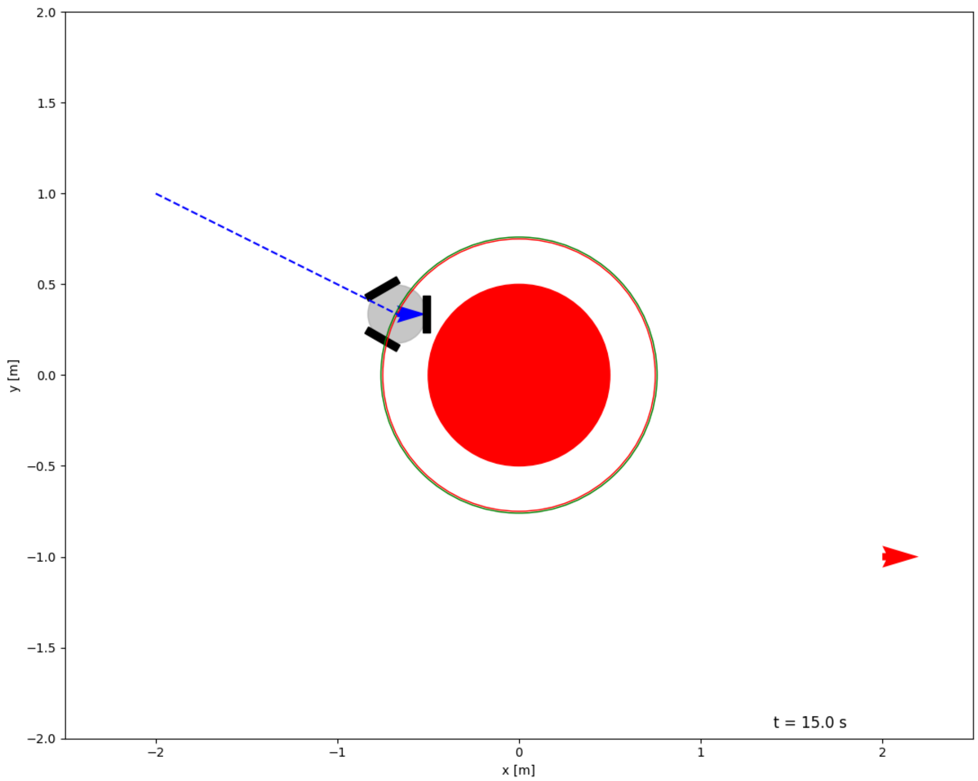

Figure 7: deadlock situation

The robot approachs the d_safe boundary at a point. The point, the initial position and the goal position became a straight line, so the robot is stuck at the point and two control input were two opposite vectors, so the robot motion was going forware and backware until the time ends.

Problem 2

In this problem, omnidirectional mobile robot was used with the position (p_x, p_y)

$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix}$$

And the control input:

$$u = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

Initial position $x[0] = [-2 \;\; 1 \;\; 0]^T$

The goal $x^d = [2 \;\; 0 \;\; 0]^T$

Figure 8 shows an obstacle shape in the file and the robot's sensor ring which has a range 1 m

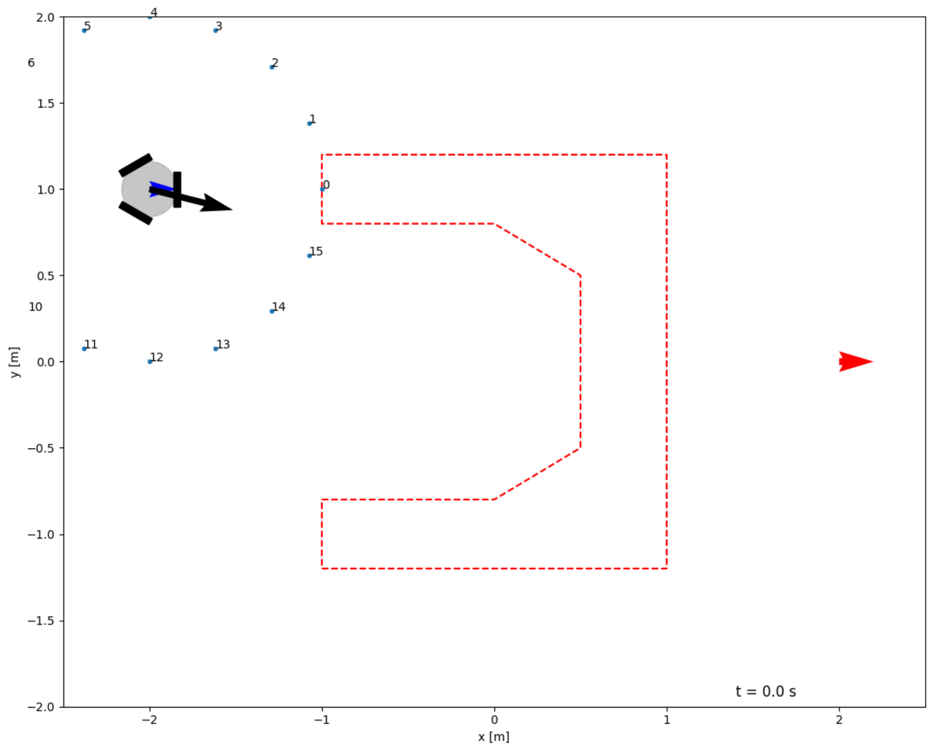

Figure 8: The robot and the obstacle and the goal

The purpose of the task is to reach the goal while avoiding collision with the obstacle.

The ring around the robot has 16 numbers which mean 16 sensors. From those sensors, there are 2 data which is needed in this problem. The first one is distance_reading, it has 16 elements, the maximum is 1, which means the sensor does not approach obstacle, the smaller value means the near the robot to the obstacle. The second one is obst_points, it is a matrix 2x16, which means the first array is x-coordinate and the second is y-coordinate of those sensors.

For this problem, the robot can be controlled by four control input: go-to-goal, avoidance, wall-following clockwise and wall-following counter clockwise. The robot started with u_gtg, when it approaches obstacle, it changed to u_wf, while the robot under u_wf, it can be deviate and cause collision with the obstacle, so u_avo will avoid it. Then the robot changed back to u_wf, until the path is clear to change to control input u_gtg.

Control input go-to-goal u_gtg

For u_gtg control input, the proportional control k_g was designed based on the equation

$$k_g = \frac{v_0\left(1 - e^{-\beta\|\bar{e}\|}\right)}{\|\bar{e}\|}$$

With $\|\bar{e}\|$ is the magnitude of the error between the desired state and the current state

$$\|\bar{e}\| = \theta_d - \theta$$

Dong Le

v_0 was set to 3, and β was set to 0.4. The value of k_g was depended on the value of v_0 and β.

Control input avoidance u_avo

For u_avo control input, the proportional control k_o was designed based on the equation

$$k_o = \frac{1}{\|\bar{e}\|}\left(\frac{c}{\|\bar{e}\|^2 - eps}\right), c > 0$$

With c was set to 5 and $\|\bar{e}\|$ is the magnitude of the error between the current state and the obstacle state

$$\|\bar{e}\| = \theta - \theta_o$$

θ_o was defined by distance_reading, at first, find out which sensor has a small value, which means it is nearest to the obstacle. Then find the index of its sensor and from that index, take the coordinate of the sensor by looking for at obst_points.

Computing x_o

The most importance part of this method was to find two sensors nearest to the obstacle. The robot can use control input u_wf clockwise or counter clockwise, with two controllers, it has different logic to find two sensors to compute u_wf_cc or u_wf_c.

For u_wf_cc sensors, firstly, find the smallest distance_reading, then check whether the sensor is less than 1, which means it has already approached the obstacle. Then find two neighbors of its sensor, if the neighbor has the value less than 1, it was accepted.

Secondly, divided into 3 situations, x1 is the smallest sensor, x2 is its neighbor when x1's index increases by 1, and x3 is its neighbor when x1's index decreases by 1.

Situation 1: all three sensors are accepted.

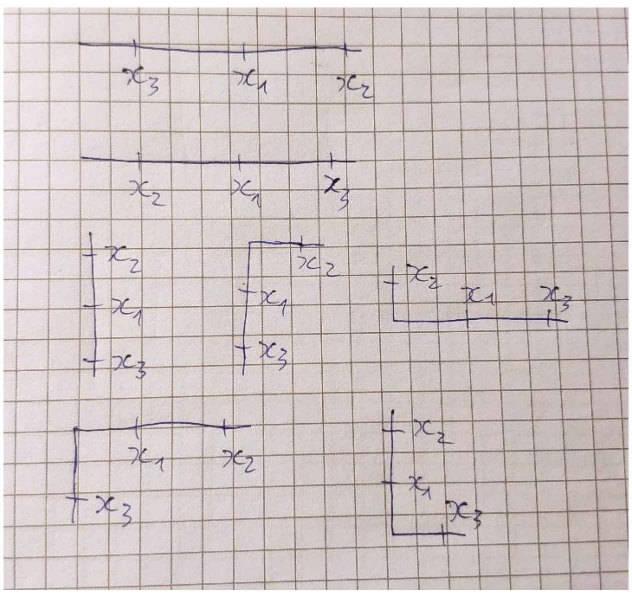

Figure 9: possible scenarios with 3 sensors

Dong Le

Figure 9 shows all scenarios of 3 sensors, the first one, choose x_o_1 is x2, x_o_2 is x3. Second one, x_o_1 is x2, x_o_2 is x3. Third one, x_o_1 is x2, x_o_2 is x3. Fourth, x_o_1 is x1, x_o_2 is x3. Fifth, x_o_1 is x1, x_o_2 is x3. Sixth, x_o_1 is x2, x_o_2 is x1. Seventh, x_o_1 is x2, x_o_2 is x1. Other than these cases, compare x3 – x2 to decide x_o_1 and x_o_2.

Situation 2 and 3: either x2 or x3 exist, so choosing x1 and x2 or x3 to be x_o_1 and x_o_2.

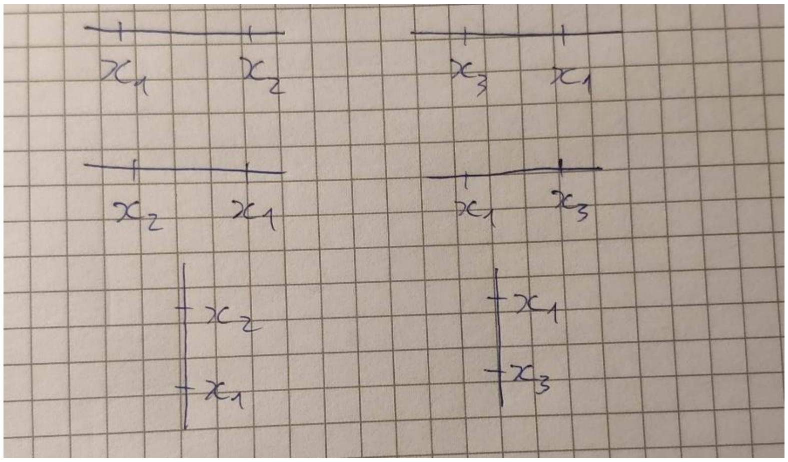

Figure 10: Possible scenarios of 2 sensors

Figure 10 shows possible scenarios of either x2 or x3 exist, along with x1 to choose x_o_1 and x_o_2. If both sensors have same x-coordinate in the figure 9, 2 sensors are vertical, then which one have smaller y-axis is x_o_2 and bigger y-axis is x_o_1. If both sensors have same y-coordinate, in first line, the sensor in the left is x_o_2 and the right is x_o_1, and in second line, the sensor in the right is x_o_2 and the left is x_o_1.

For u_wf_c sensors, it is quite the same with the u_wf_cc sensors, but it needs to be opposite of choosing x_o_1 and x_o_2.

Control input wall-following counter clockwise u_wf_cc

Firstly, compute a vector tangential to the wall

$$u_{wf,t} = x_{o2} - x_{o1}$$

$$\bar{u}_{wf,t} = \frac{u_{wf,t}}{\|u_{wf,t}\|}$$

Secondly, compute a vector perpendicular to the wall

$$u_{wf,p} = (x_{o1} - x) - \left((x_{o1} - x) * \bar{u}_{wf,t}\right) * \bar{u}_{wf,t}$$

$$\hat{u}_{wf,p} = u_{wf,p} - \frac{d^{des}}{\|u_{wf,p}\|} * u_{wf,p}$$

With d^{des} is desired distance from the wall, in this case choosing $d^{des} = d\_safe$

Finally, combine two vectors.

$$u_{wf}^{cc} = \hat{u}_{wf,p} + \bar{u}_{wf,t}$$

Control input wall-following clockwise u_wf_c

Firstly, compute a vector tangential to the wall.

$$u_{wf,t} = x_{o1} - x_{o2}$$

$$\bar{u}_{wf,t} = \frac{u_{wf,t}}{\|u_{wf,t}\|}$$

Secondly, compute a vector perpendicular to the wall.

$$u_{wf,p} = (x_{o2} - x) - \left((x_{o1} - x) * \bar{u}_{wf,t}\right) * \bar{u}_{wf,t}$$

$$\hat{u}_{wf,p} = u_{wf,p} - \frac{d^{des}}{\|u_{wf,p}\|} * u_{wf,p}$$

With d^{des} is desired distance from the wall, in this case choosing $d^{des} = d\_safe$

Finally, combine two vectors.

$$u_{wf}^{c} = \hat{u}_{wf,p} + \bar{u}_{wf,t}$$

From u_gtg, u_avo, u_wf_cc, and u_wf_c, a new control input was designed. Figure 11 shows the robot's trajectory to the goal

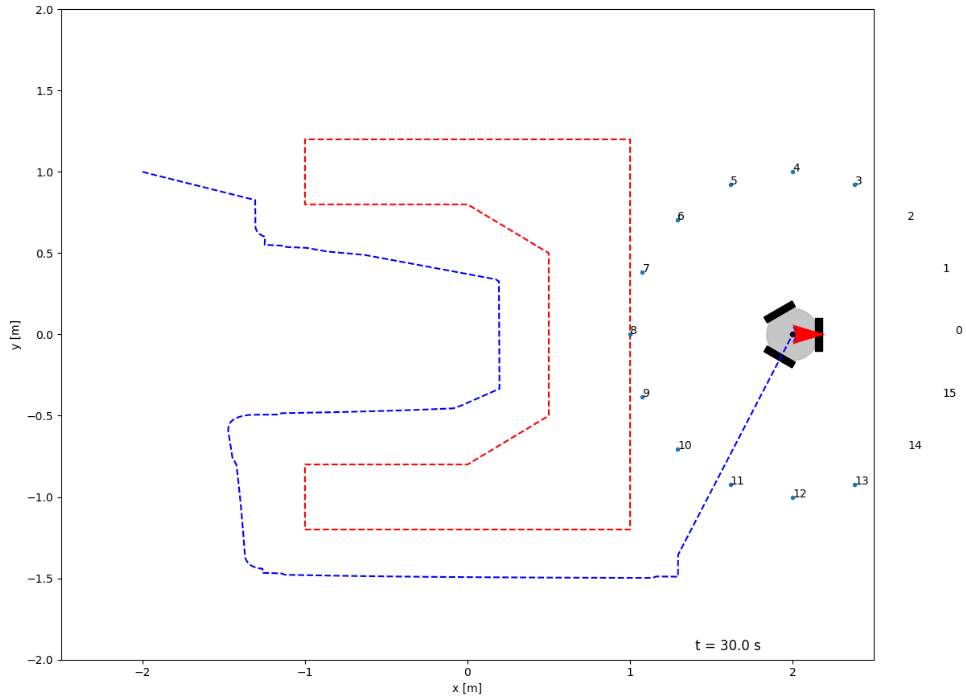

Figure 11: Robot's trajectory to the goal

The figure 12 shows the control input and the speed of the robot, thanks to the saturation, the speed of the robot is always less or equal than 0.5

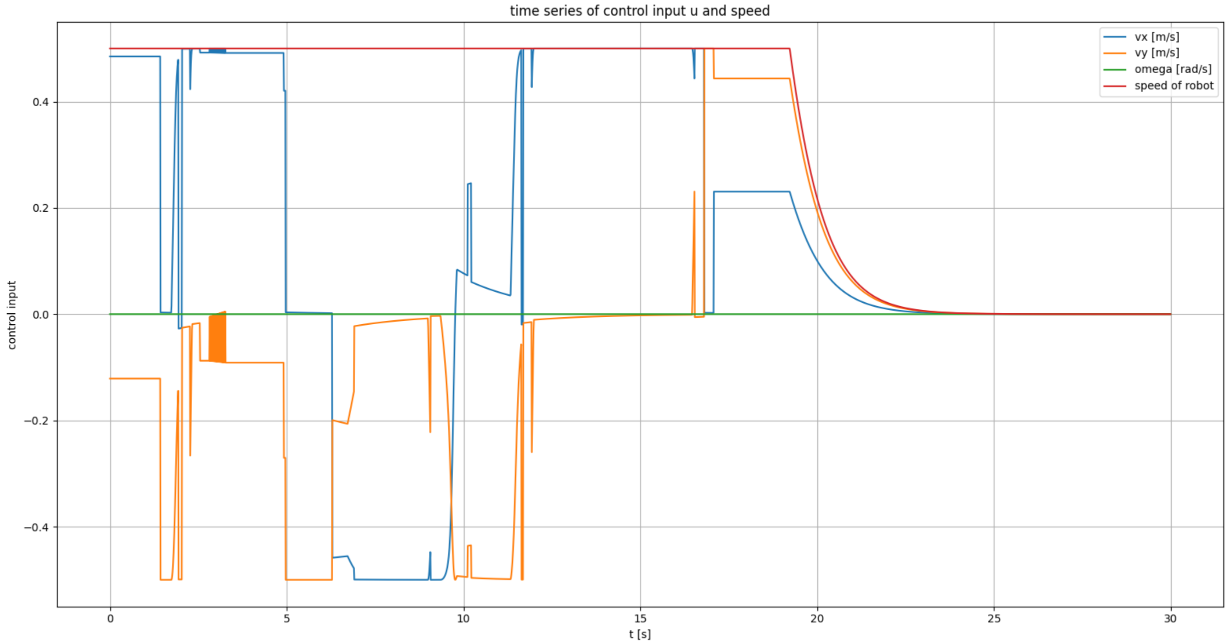

Figure 12: Control input and speed

Figure 13 shows the error between the goal state and robot state, so after more than 20s, the robot reaches the goal, so the errors became 0

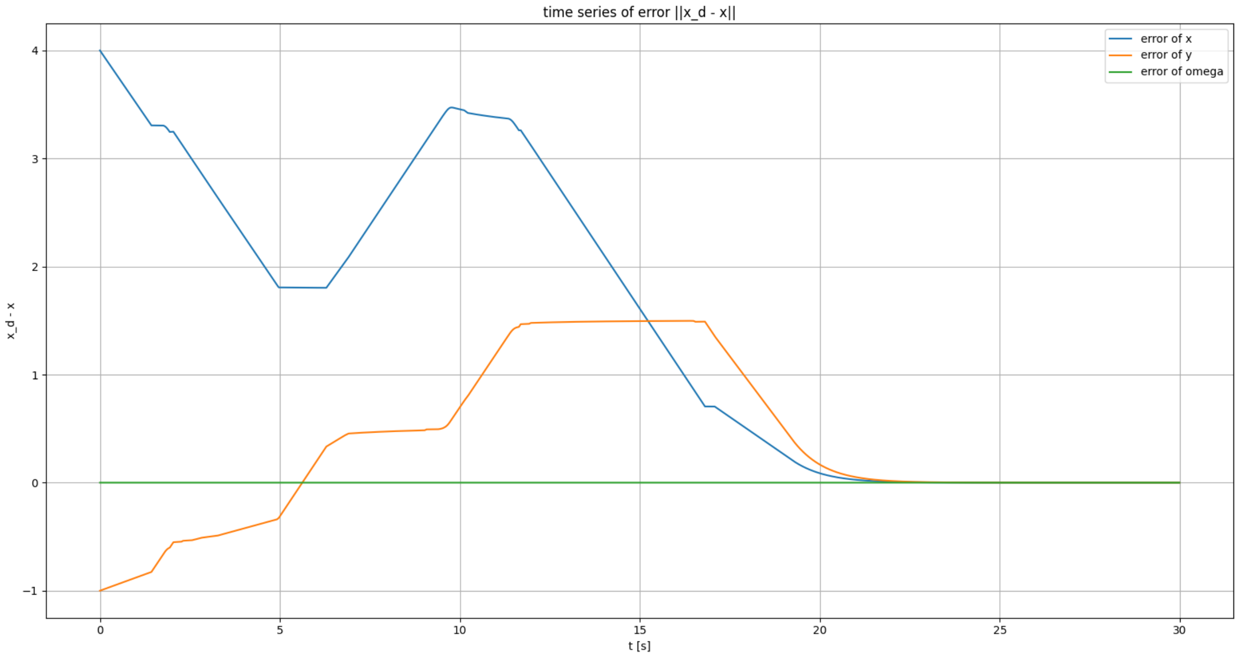

Figure 13: Error between x_d and x

Figure 14 shows the minimum reading distance from the sensor.

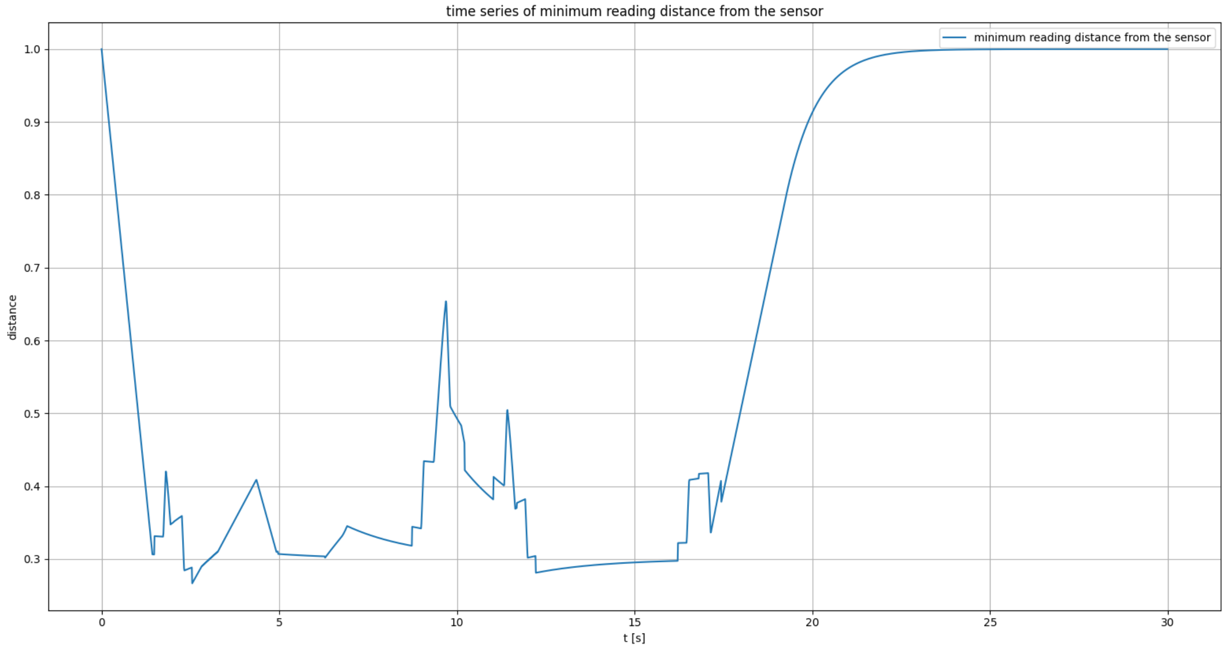

Figure 14: Minimum reading distance from the sensor

Figure 15 shows the goal state and the robot state, so when the robot reaches the goal, the robot's state became the goal's state.

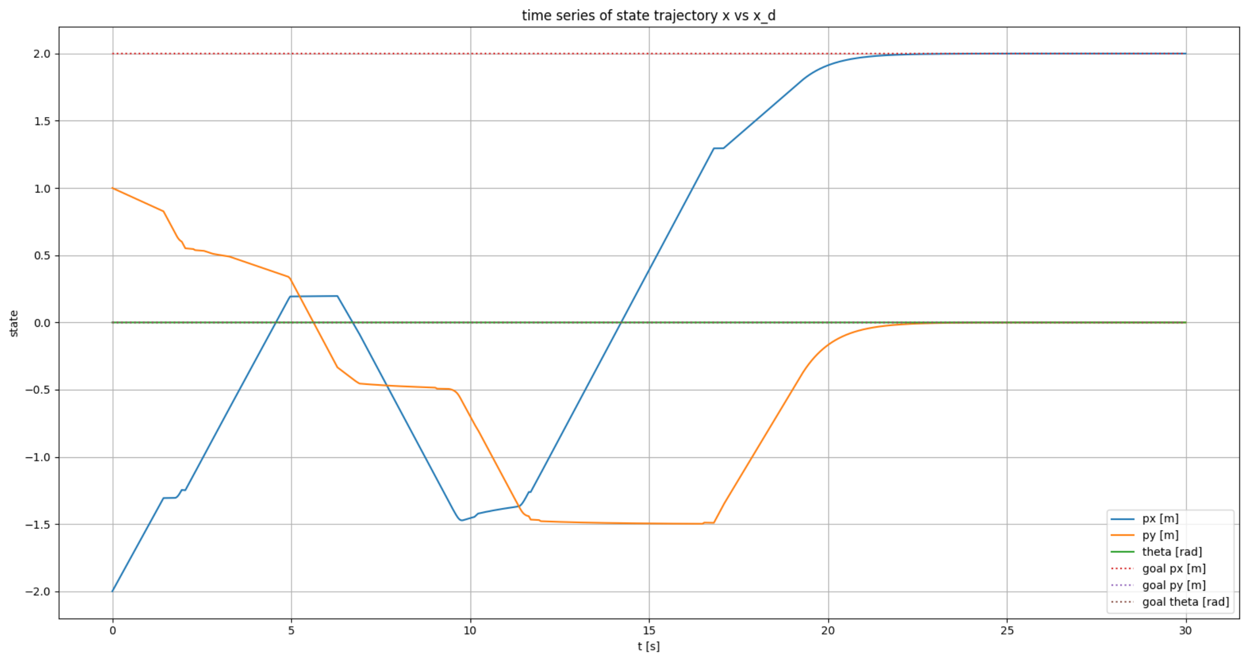

Figure 15: The state trajectory x and x_d

Dong Le

In conclusion, by using the wall-following method, the robot can overcome un-defined shape obstacles, changing the control input based on the condition and previous control state helps the ride smoother than the hard-switching in the problem 1, and it also needs less time than hard-switching.

During the trajectory, when u_gtg and u_avo became symmetric, the robot does not stop or stuck there, it continues with the previous control state as wall-following counter clockwise, so it is more optimized than hard-switching.

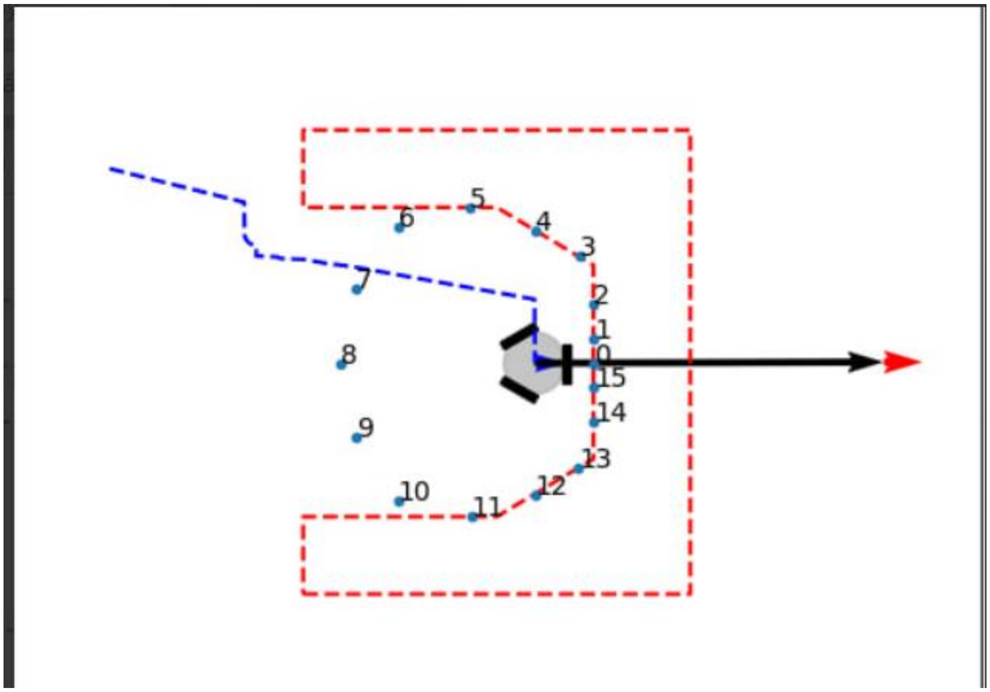

Figure 16: u_gtg, u_avo, robot state become symmetric.

Problem 3

In this problem, omnidirectional mobile robot was used with the position (p_x, p_y)

$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix} \quad (1)$$

And the control input:

$$u = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \quad (2)$$

Initial position $x[0] = [-2 \ -1 \ \ 0]^T$

The goal $x^d = [2 \ \ 1 \ \ 0]^T$

There are 3 circular obstacles which are:

1. Obstacle 1 is centered at $p_x^1 = -0.85$, $p_y^1 = 0$ with radius 0.3m.
2. Obstacle 2 is centered at $p_x^2 = 0.3$, $p_y^2 = -0.1$ with radius 0.3m.
3. Obstacle 3 is centered at $p_x^3 = 1.05$, $p_y^3 = 0.8$ with radius 0.3m.

Figure 17 shows the three obstacles, the robot initial position and the goal.

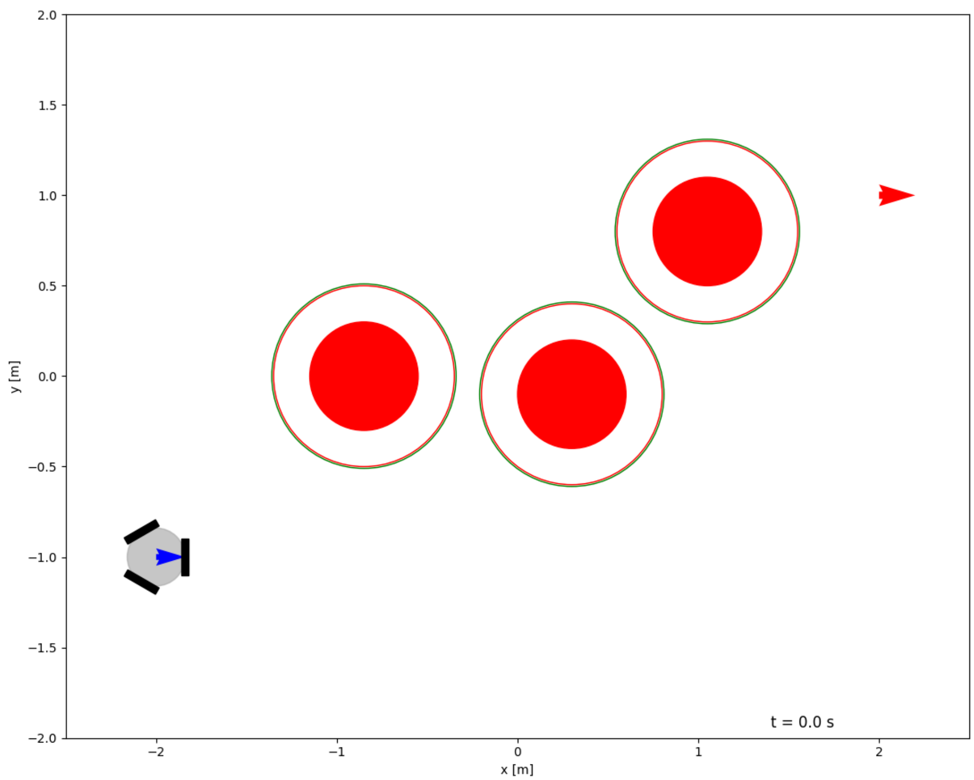

Figure 17: The robot and three obstacles

The purpose of the task is to reach the goal while avoiding collision with obstacle.

At first, a control input go-to-go was designed. For u_gtg control input, the proportional control k_g was designed based on the equation

$$k_g = \frac{v_0\left(1 - e^{-\beta\|\bar{e}\|}\right)}{\|\bar{e}\|} \tag{5}$$

With $\|\bar{e}\|$ is the magnitude of the error between the desired state and the current state

$$\|\bar{e}\| = \theta_d - \theta \tag{6}$$

v_0 was set to 3, and β was set to 0.4. The value of k_g was depended on the value of v_0 and β.

To ensure that the designed control input satisfy the robot maximum speed, the go-to-go control input must be reduced to $\sqrt{(v_x^2 + v_y^2)} = 0.5\, m/s$

Therefore, after calculating the go-to-goal control input, the speed v_x and v_y became.

$$v_{x\_gtg} = \frac{v_{x\_gtg} * 0.5}{\sqrt{\left(v_{x\_gtg}^2 + v_{y\_gtg}^2\right)}} \tag{9}$$

$$v_{y\_gtg} = \frac{v_{y\_gtg} * 0.5}{\sqrt{\left(v_{x\_gtg}^2 + v_{y\_gtg}^2\right)}} \tag{10}$$

Secondly, Implement the QP-based controller as follows:

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

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.

From the QP-based controller, it was converted to:

$$\begin{aligned} \min_u \; & \|u_{gtg} - u\|^2 \\ \text{s.t.} \quad & -2(x - x_o)^T u \leq \gamma(h(x)) \end{aligned} \quad \overset{z = u}{\Longleftrightarrow} \quad \begin{aligned} \min_z \; & \frac{1}{2} z^T Q z + c^T z \\ \text{s.t.} \quad & Hz \preceq b \end{aligned}$$

In this problem :

$$Q = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$c = -2 * \begin{bmatrix} u\_gtg\_x \\ u\_gtg\_y \end{bmatrix}$$

$$H = -2 * \begin{bmatrix} p_x - p_x^1 & p_y - p_y^1 \\ p_x - p_x^2 & p_y - p_y^2 \\ p_x - p_x^3 & p_y - p_y^3 \end{bmatrix}$$

For $b = \gamma(h) = 0.2h$

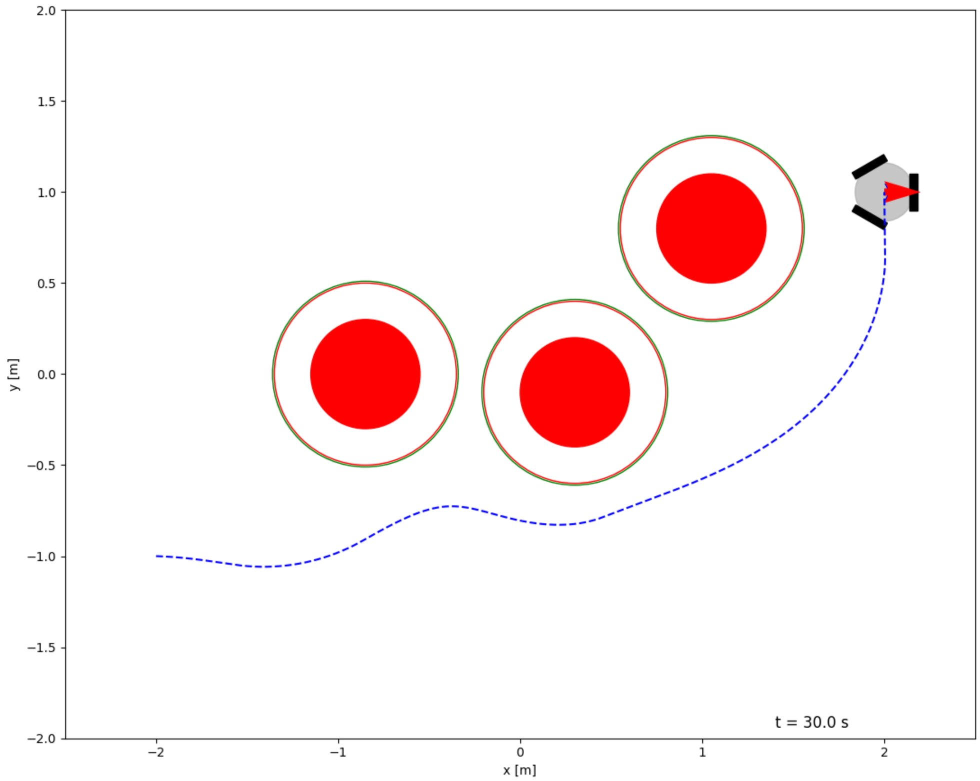

Figure 18: Robot trajectory with y(h) = 0.2h

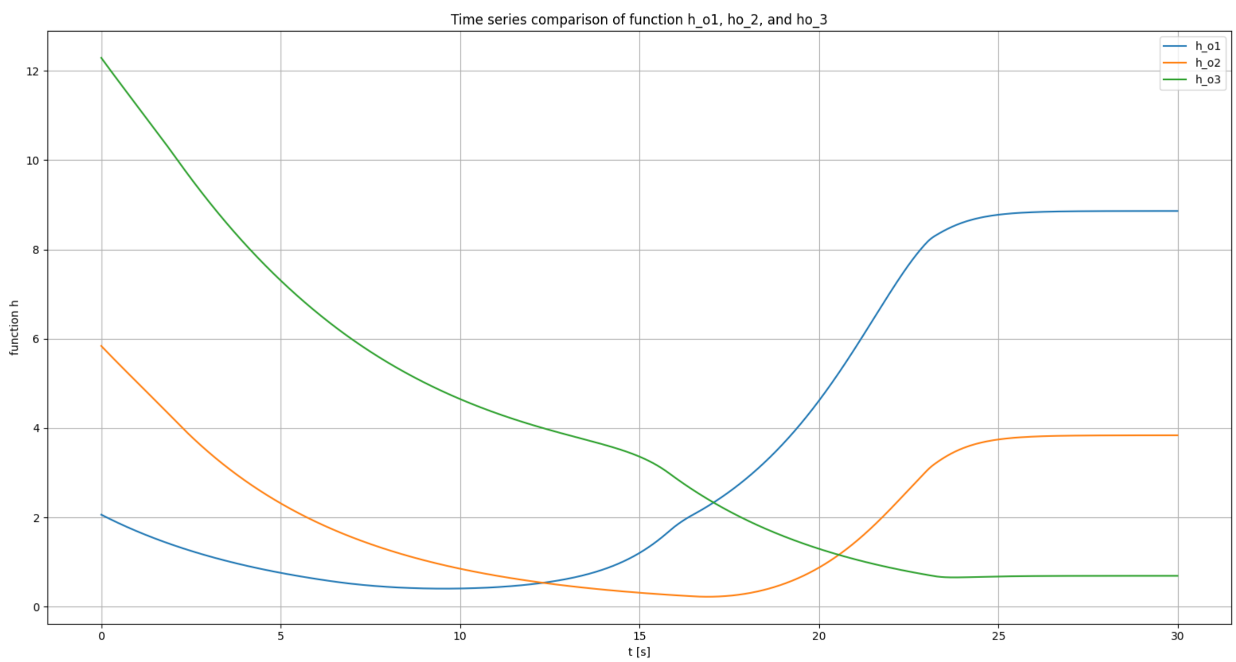

Figure 19: Distance from robot to obstacles with y(h) = 0.2h

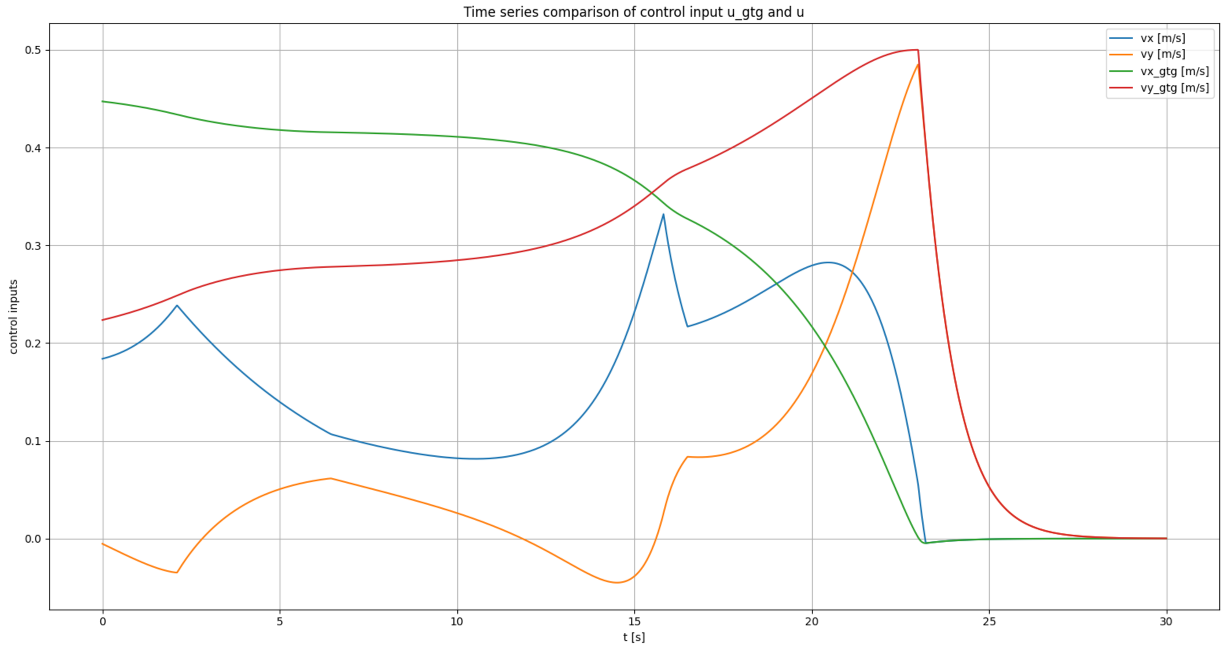

Figure 20: Comparation between control input and go-to-goal with y(h) = 0.2h

For $b = \gamma(h) = 10h$

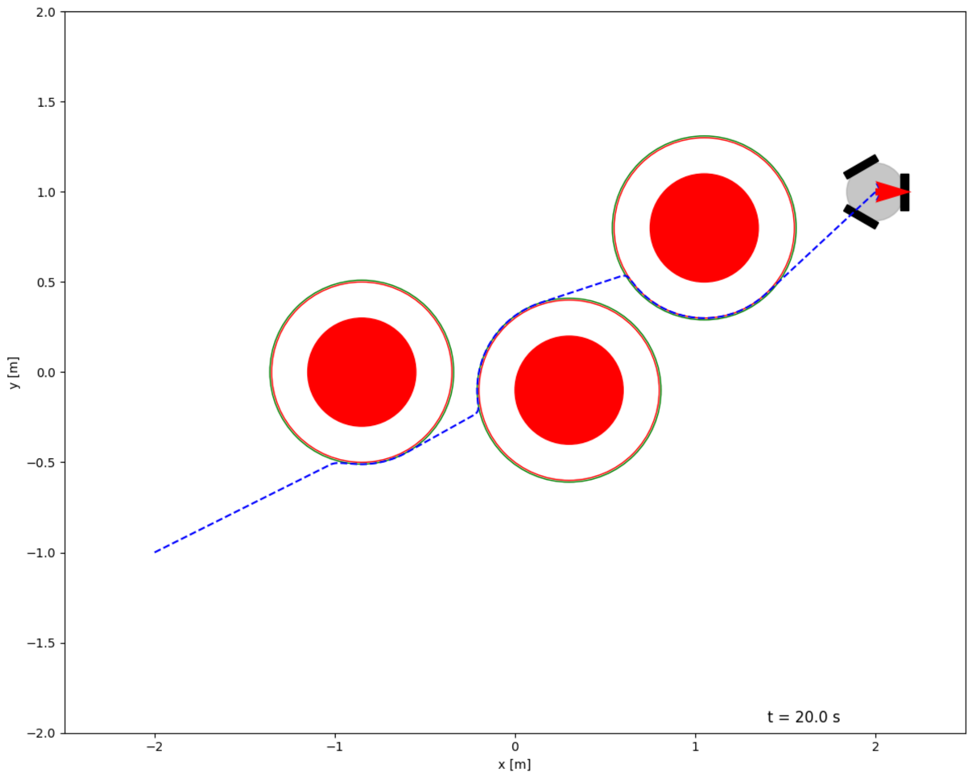

Figure 21: Robot trajectory with y(h) = 10h

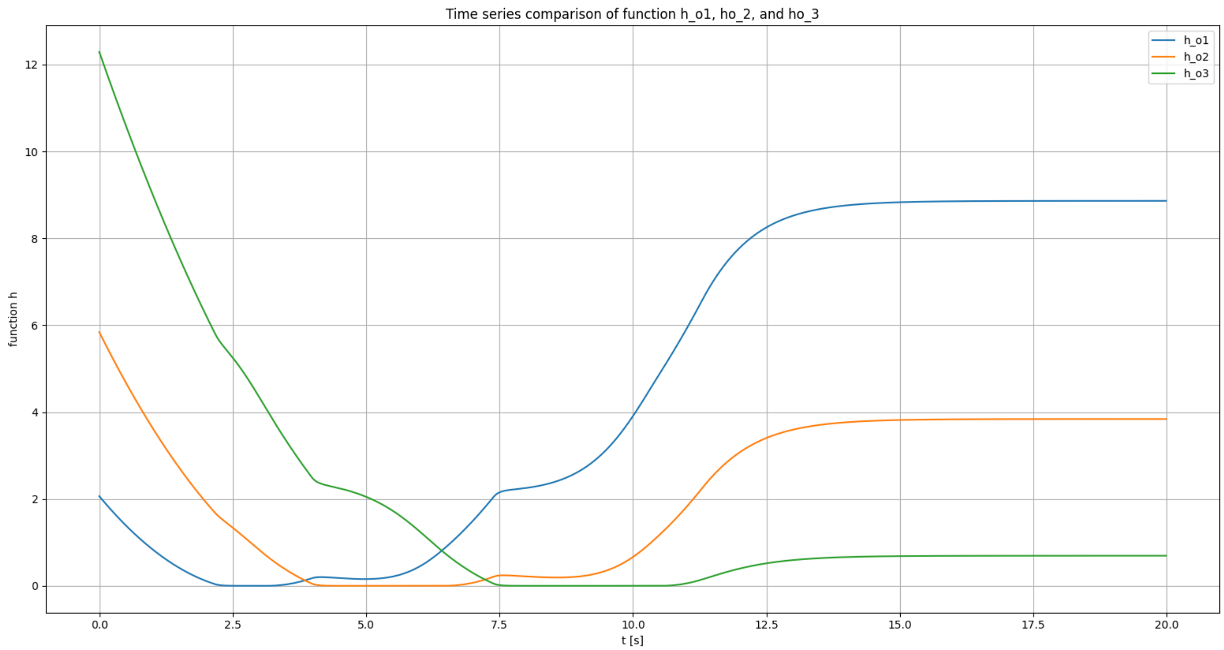

Figure 22: Distance from robot to obstacles with y(h) = 10h

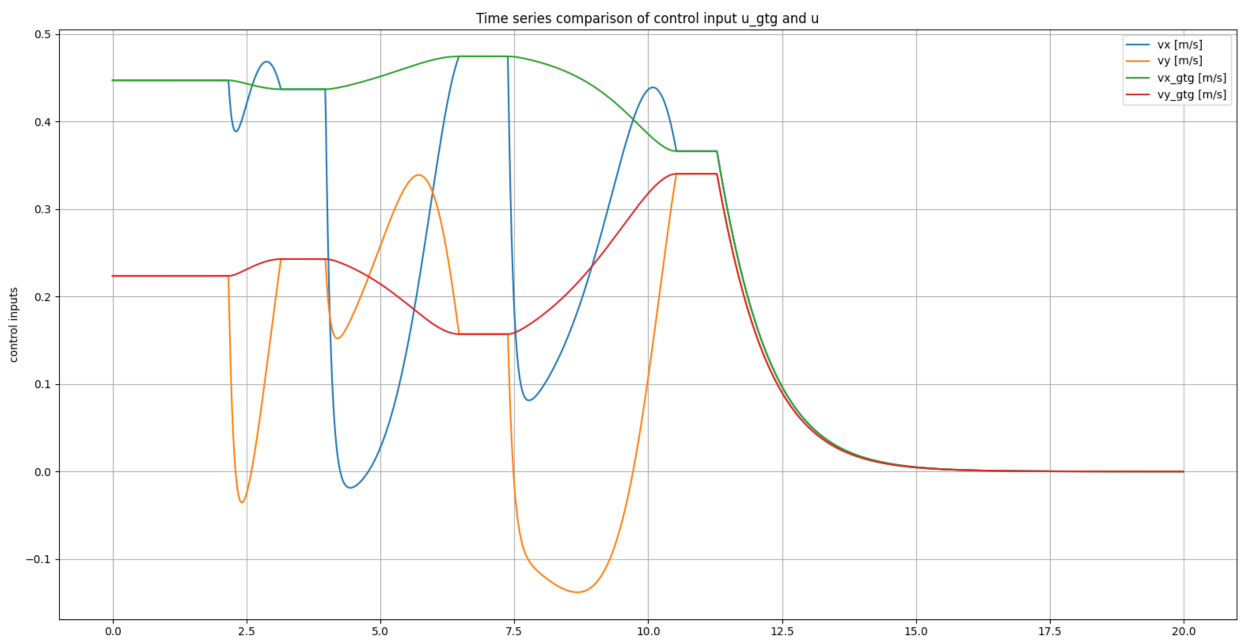

Figure 23: Comparation between control input and go-to-goal with y(h) = 10h

For $b = \gamma(h) = 10h^3$

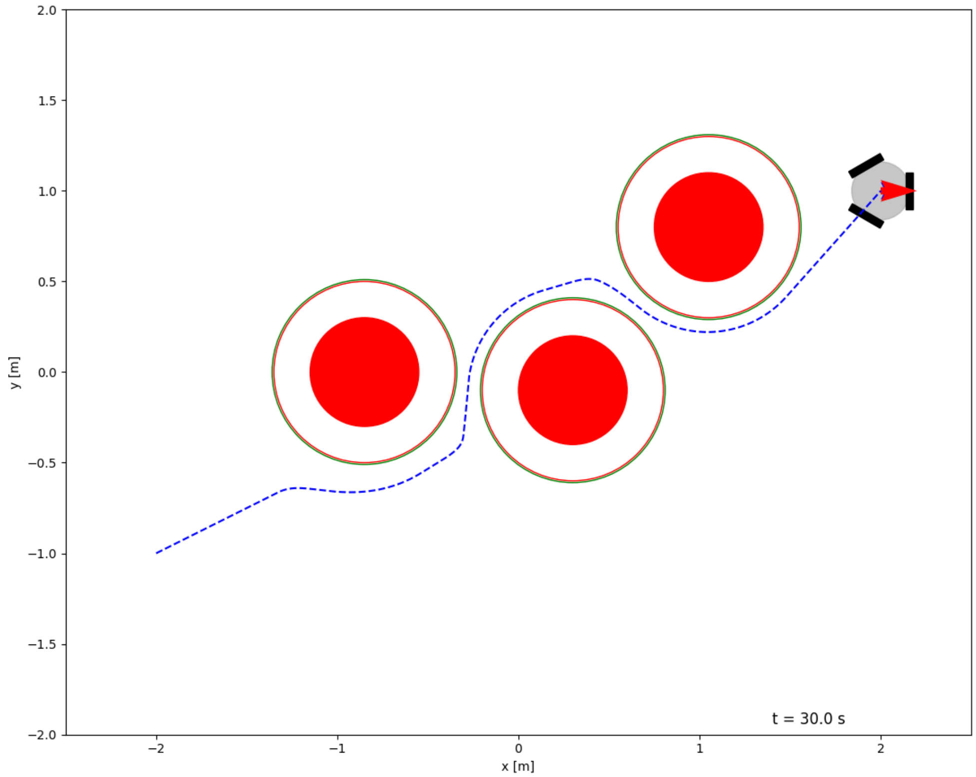

Figure 24: Robot trajectory with y(h) = 10h^3

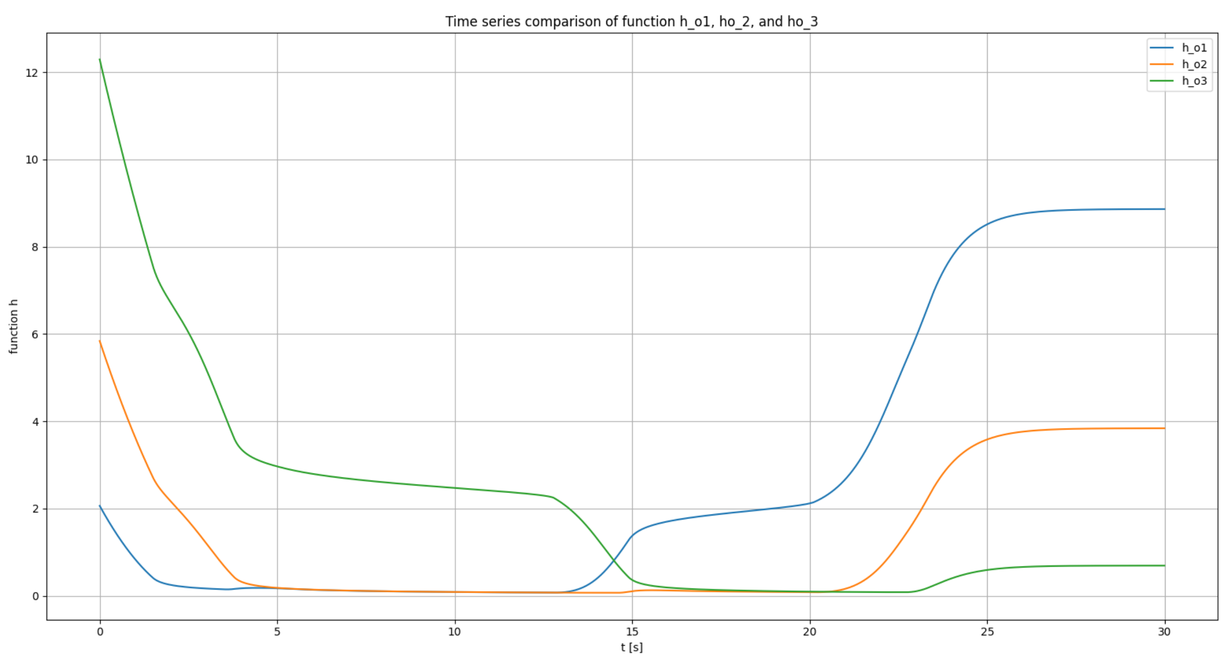

Figure 25: Distance from robot to obstacles with y(h) = 10h^3

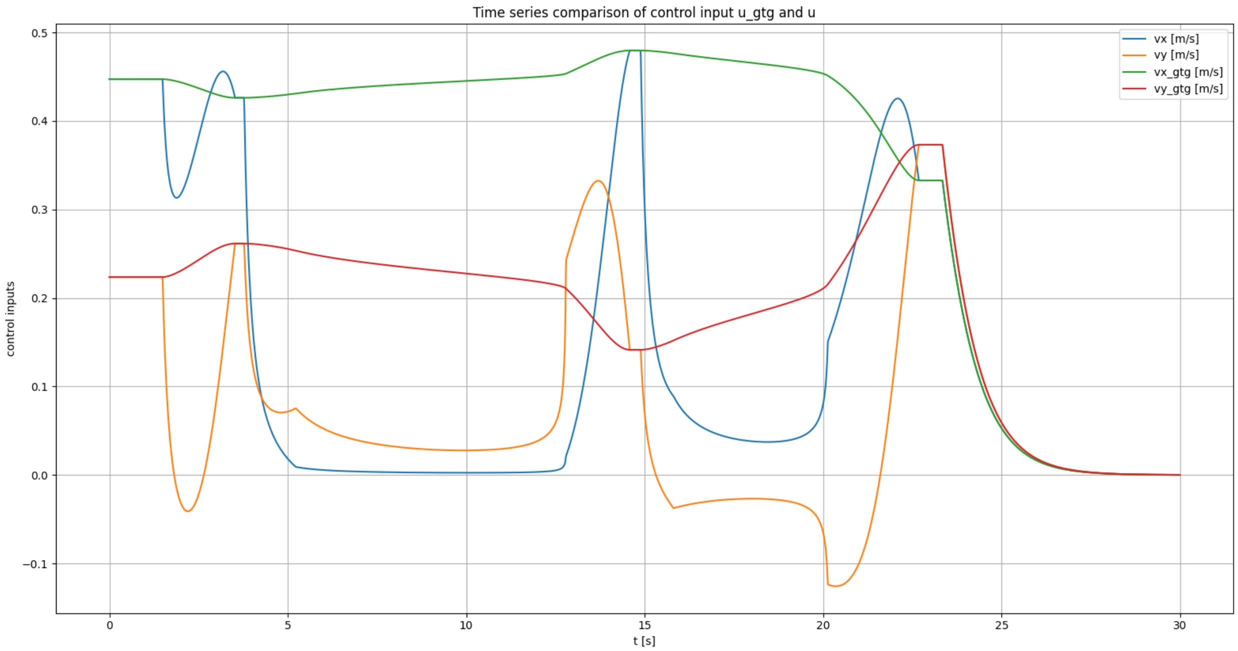

Figure 26: Comparation between control input and go-to-goal with y(h) = 10h^3

Comparation

It can be clearly seen that the safety filter reaction to an obstacle depend on the selection of γ function. If γ gets smaller, the safety controller will be activated way before reaching the safe set boundary, the controller will prioritize collision avoidance over other go-to-goal objectives. This behavior can be seen by above graphs where the safety filter reacts to the obstacle early and deviate easily from the go-to-goal when the γ has a lower value $\boldsymbol{\gamma(h) = 0.2h}$

So, what should be selected as the best out of three depends on the scenario. For example, if the noise of the measurements is higher, then we must keep higher weight for the collision avoidance and need the safety filter to be reacted much aggressively. On the other hand, if the parameters are well defined, we can provide more weight on the go-to-goal objective. However, considering a generic situation, $\boldsymbol{\gamma(h) = 10h^3}$ can be considered as a better option as it makes sure enough safety while not reacting too early.

Dong Le

Conclusion

By implementing the first problem, the result found out that with the hard switching method between go-to-goal and avoidance can cause a non-smooth path of the robot to reach the goal. The choice of d_{safe} and eps affects the condition to switch, therefore, carefully consider choosing d_{safe} and eps.

In the second problem the result found out that applying wall-following can guide the robot overcome unique shape obstacle, and choosing sensors to compute the control input wall following is important, otherwise, the robot’s behaviour can be different and out-of-control.

In the third problem the result found out by applying quadratic programming, an optimized control input was designed. Also, the value of $\gamma(h)$ affects the robot’s behaviour.

Moreover, in all three problems, the speed of robot needs to saturate to ensure robot’s limitation.

Dong Le

Appendices

Appendix A

Program code is attached to the zip file.

THIS PAGE LEFT BLANK INTENTIONALLY