UNIVERSITÀ
DEGLI STUDI
DI PADOVA

The PAC-MAN Challenge

Scantamburlo Mattia
Cozza Mattia
Piai Luca
Chinello Alessandro




Project Overview

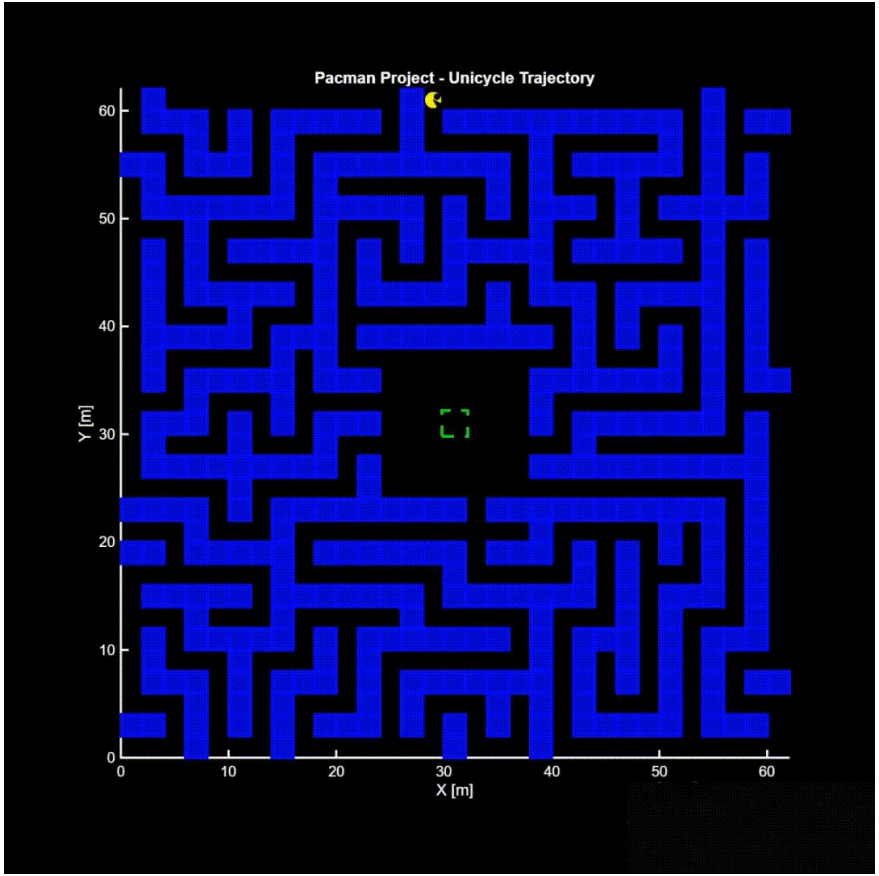
Goal: Determine which controller demonstrates the highest generalization capability.

Methodology:

- **Tuning:** All controllers were tuned using the identical set of scenarios.
- **Testing:** Each controller's performance was evaluated on a pre-defined path within the Pac-Man maze environment.

Link to the repository:

https://github.com/robotics-group10/PAC-MAN



Project Overview

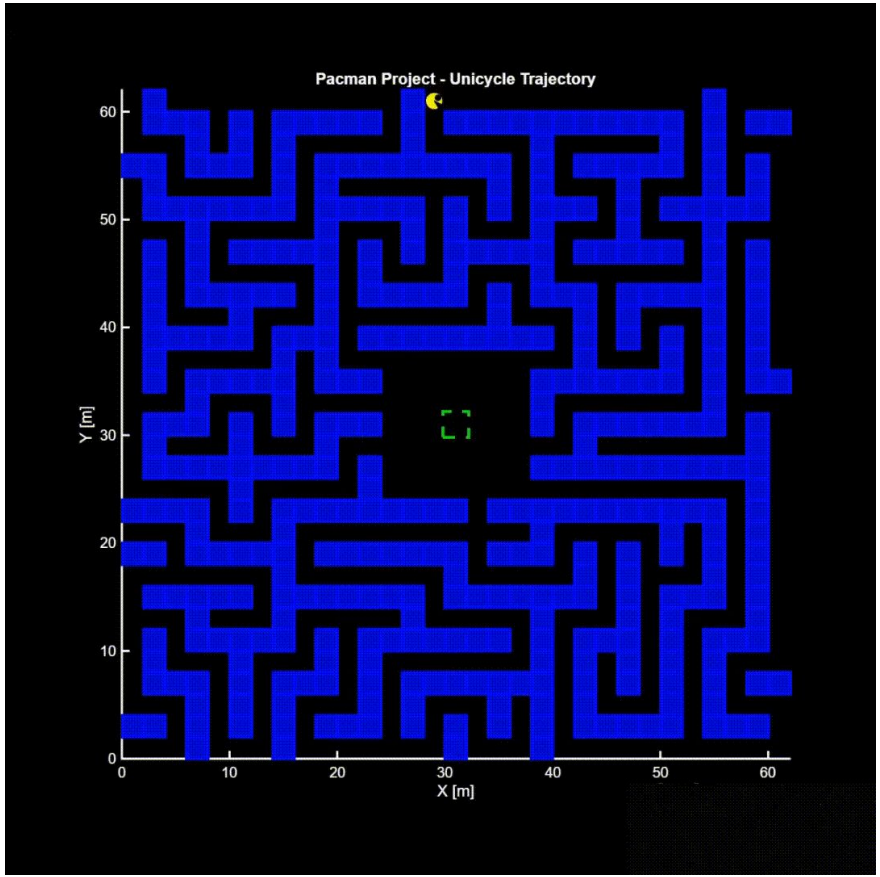
The project is divided in 4 main parts:

1. Define the **maze** with the **parking area** and the **path planning** logic

2. Define simple scenarios for tuning both classes of controllers (**trajectory tracking** and **regulation**)

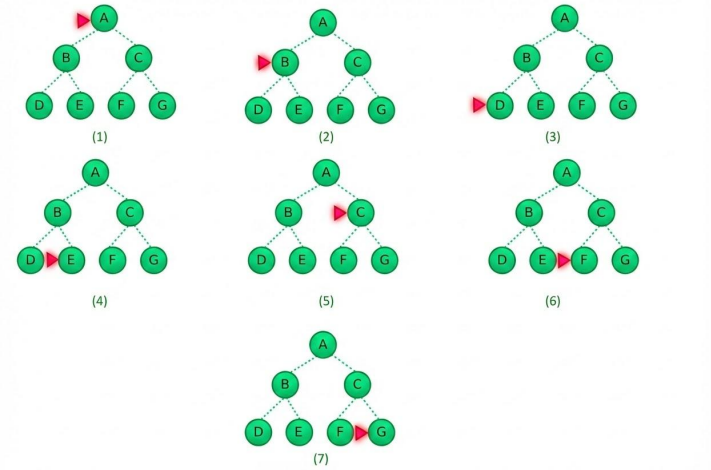
3. Perform **tuning** using a **grid search algorithm** to automatize the simulations

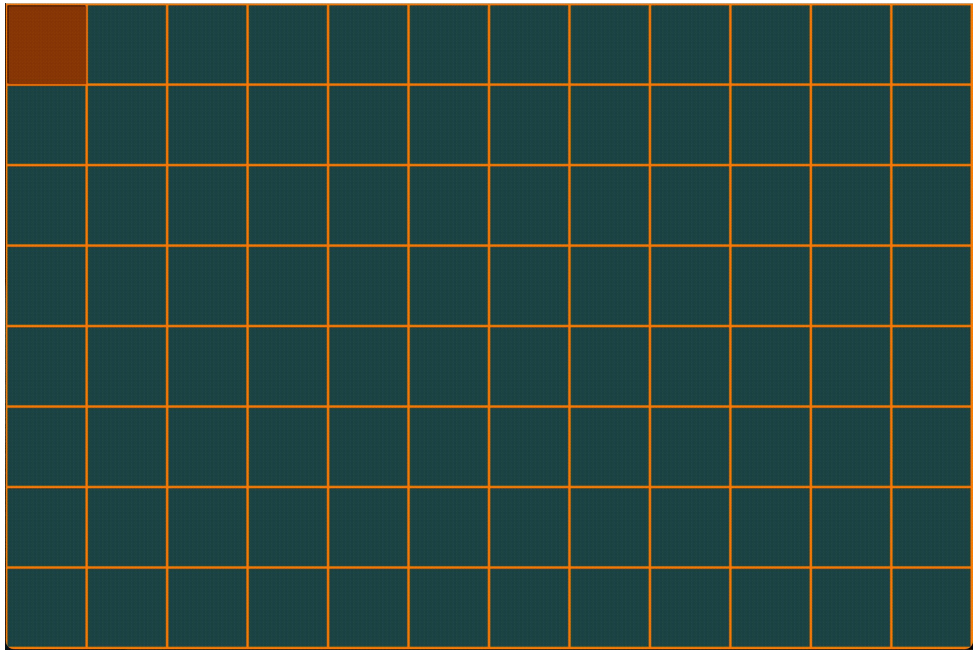
4. Run on the maze the controllers applied on the **unicycle model** with the best params found



Maze Generation

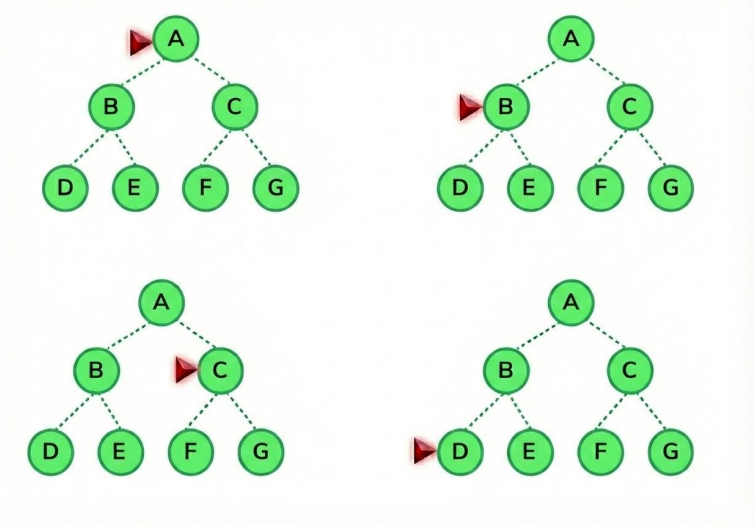
A deterministic randomized **Depth-First Search** algorithm generates a maze.
A **central square room** is subsequently carved to define the regulation target region.

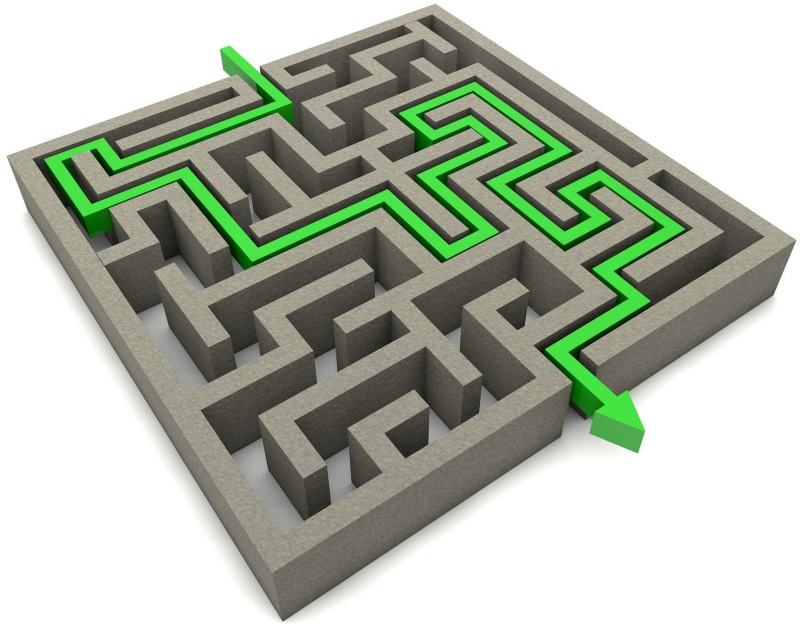




Path Planning Logic

A **Breadth-First Search** algorithm explores the maze from the start position to the goal, marking visited cells and backtracking to reconstruct the shortest path.
Then a subsequent **interpolation** phase to obtain a continuous path.





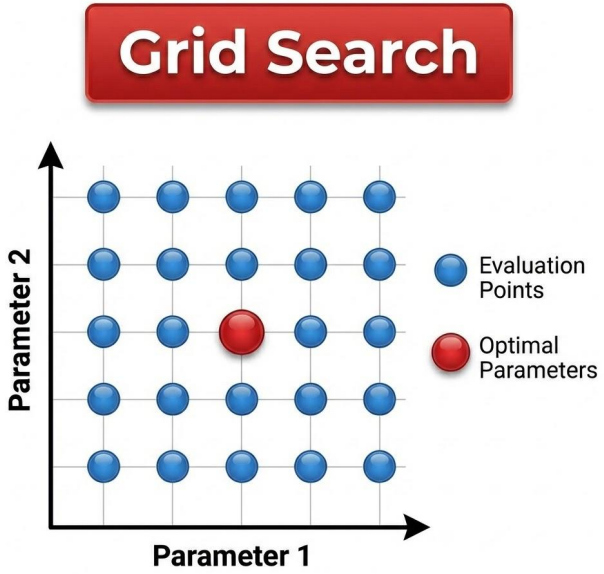
Tuning

The grid search algorithm

We implemented a **Grid Search algorithm** focused on the domain of gains and parameters where convergence is theoretically guaranteed within a finite set of values.

This systematic exploration was conducted to empirically evaluate the performance of each configuration, aiming to identify the specific parameter sets that yield **the lowest error based on the specified cost function**.

The aim doing this is to select the parameter that in the simulation gets on average the **fastest convergence**.



The tracking and parking cost functions quantify performance by computing two different errors:

- **Tracking cost** measures the **mean Euclidean distance** between actual and desired trajectories

$$e(t) = \|q(t) - q_d(t)\|_2 = \sqrt{(x(t) - x_d(t))^2 + (y(t) - y_d(t))^2}$$

$$J_{\text{tracking}} = \frac{1}{T} \int_0^T e(t)\, dt$$

- **Parking cost integrates squared distance** to the goal and **lightly penalizes control effort** over time

$$\rho(t) = \sqrt{(x(t) - x_d)^2 + (y(t) - y_d)^2}$$

$$J_{\text{parking}} = \frac{1}{\|[x_d, y_d]\|} \int_0^T \left(\rho(t)^2 + \alpha v(t)^2 + \beta \omega(t)^2\right) dt, \quad \alpha, \beta \in [0, 1]$$

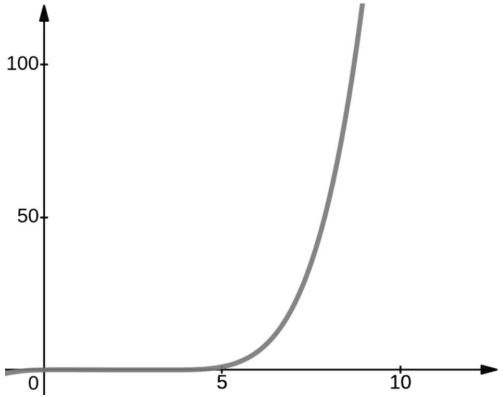
In practice tuning had to be studied for each specific controller.

For example in **non linear trajectory tracking** we had to tune the two parameters:

$$(\alpha, \xi)$$

while in other controllers (like **cartesian regulation**) we had to tune the gains associated with velocity and angular velocity:

$$(v, w)$$



Tuning

Example in practice

To better understand the amount of combination the simple and restrained method we used, consider the case of posture regulation:

- we had 7 different goal positions
- 3 parameters (k1, k2, k3)
- for each parameter 3 values
- 3 different grid search phases

This amount to about:

[7*(3^3)]*3 = **567 combinations**

(which has an exponential growth with parameters domains sizes)

```
=== Tuning Parking Goal [5.0, 0.0] ===
Tried: k1=2.00, k2=2.00, k3=2.00 -> Error: 1.9194
Tried: k1=6.00, k2=2.00, k3=2.00 -> Error: 2.1022
Tried: k1=10.00, k2=2.00, k3=2.00 -> Error: 3.0162
Tried: k1=2.00, k2=6.00, k3=2.00 -> Error: 1.9194
Tried: k1=6.00, k2=6.00, k3=2.00 -> Error: 2.1022
Tried: k1=10.00, k2=6.00, k3=2.00 -> Error: 3.0162
Tried: k1=2.00, k2=10.00, k3=2.00 -> Error: 1.9194
Tried: k1=6.00, k2=10.00, k3=2.00 -> Error: 2.1022
Tried: k1=10.00, k2=10.00, k3=2.00 -> Error: 3.0162
Tried: k1=2.00, k2=2.00, k3=6.00 -> Error: 1.9194
Tried: k1=6.00, k2=2.00, k3=6.00 -> Error: 2.1022
Tried: k1=10.00, k2=2.00, k3=6.00 -> Error: 180.4422
Tried: k1=2.00, k2=6.00, k3=6.00 -> Error: 1.9194
Tried: k1=6.00, k2=6.00, k3=6.00 -> Error: 2.1022
Tried: k1=10.00, k2=6.00, k3=6.00 -> Error: 3.0162
Tried: k1=2.00, k2=10.00, k3=6.00 -> Error: 1.9194
Tried: k1=6.00, k2=10.00, k3=6.00 -> Error: 2.1022
Tried: k1=10.00, k2=10.00, k3=6.00 -> Error: 3.0162
Tried: k1=2.00, k2=2.00, k3=10.00 -> Error: 1.9194
Tried: k1=6.00, k2=2.00, k3=10.00 -> Error: 2.1022
Tried: k1=10.00, k2=2.00, k3=10.00 -> Error: 3.0162
Tried: k1=2.00, k2=6.00, k3=10.00 -> Error: 1.9194
Tried: k1=6.00, k2=6.00, k3=10.00 -> Error: 2.1022
Tried: k1=10.00, k2=6.00, k3=10.00 -> Error: 3.0162
Tried: k1=2.00, k2=10.00, k3=10.00 -> Error: 1.9194
Tried: k1=6.00, k2=10.00, k3=10.00 -> Error: 2.1022
Tried: k1=10.00, k2=10.00, k3=10.00 -> Error: 3.0162
Goal [5.0, 0.0] -> K1=2.00, K2=6.00, K3=10.00| Err=1.9194
```

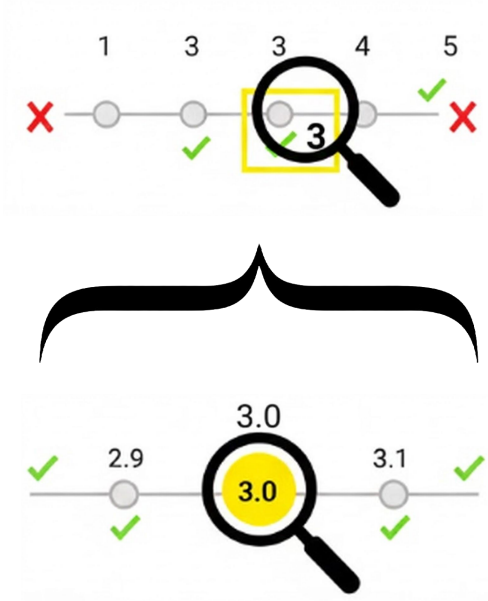
So by using the grid search we specified sets of possible values for the gains or their parametrizations.

Problems:

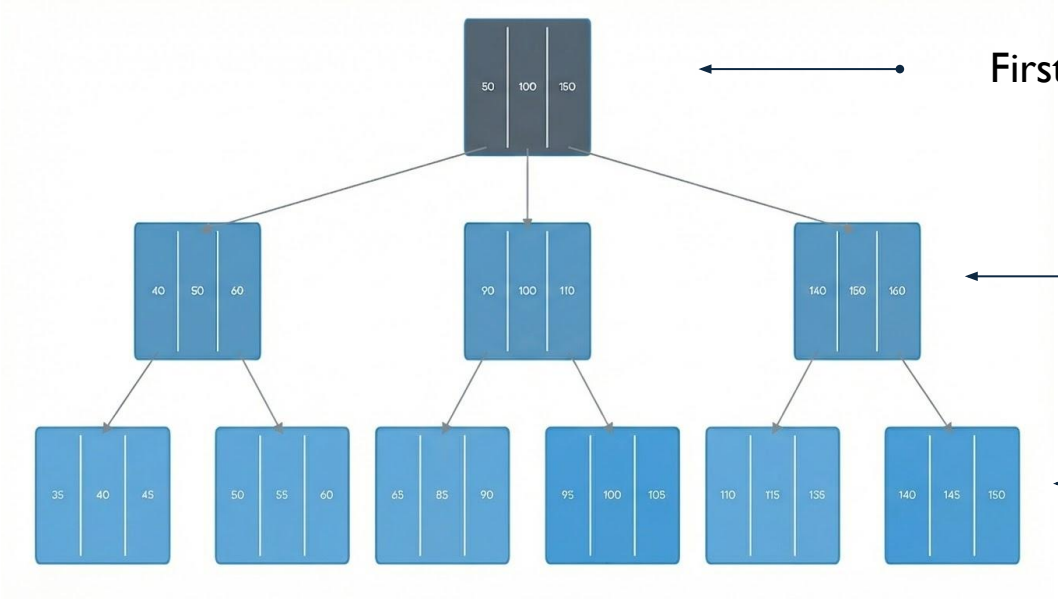
- The amount of combination to test grows **exponentially** with the size of parameters domain.

Solutions:

- One way to solve was to use **sequential grid search** with sets of values closer and closer to optimal value.



Example of multiple phases search (from large range to small range):



First level search

Second level search

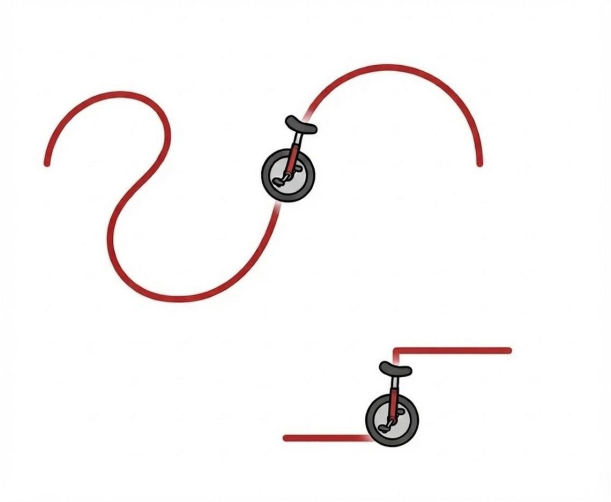
Third level search

Tuning Trajectory Controllers

The simplified scenarios

Trajectory controllers were tuned using 4 different simple trajectories (see next slides).

Output Error Feedback was also evaluated on 2 additional scenarios involving corners.



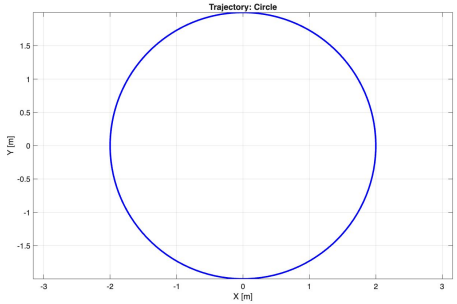
Tuning Trajectories Controller

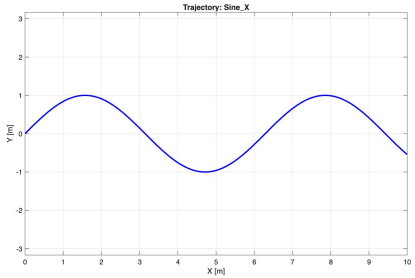
The simplified scenarios

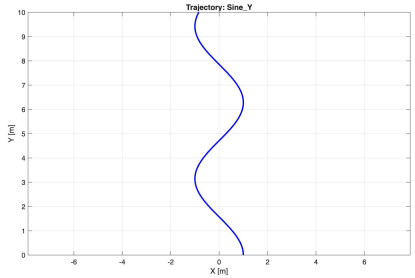
Cost function:

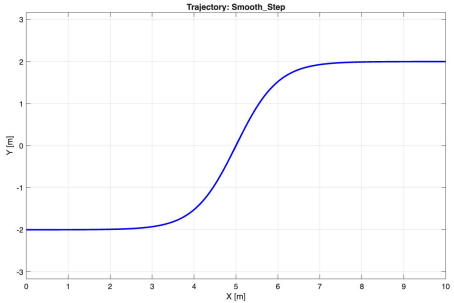
$$e(t) = \|q(t) - q_d(t)\|_2 = \sqrt{(x(t) - x_d(t))^2 + (y(t) - y_d(t))^2}$$

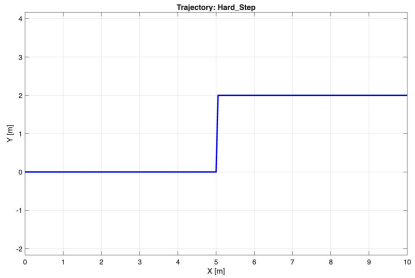
$$J_{\text{tracking}} = \frac{1}{T} \int_0^T e(t)\, dt$$



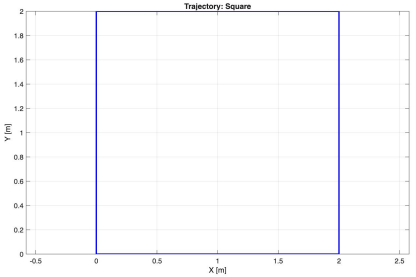








Only for 3rd controller



Only for 3rd controller

Tuning Regulation Controllers

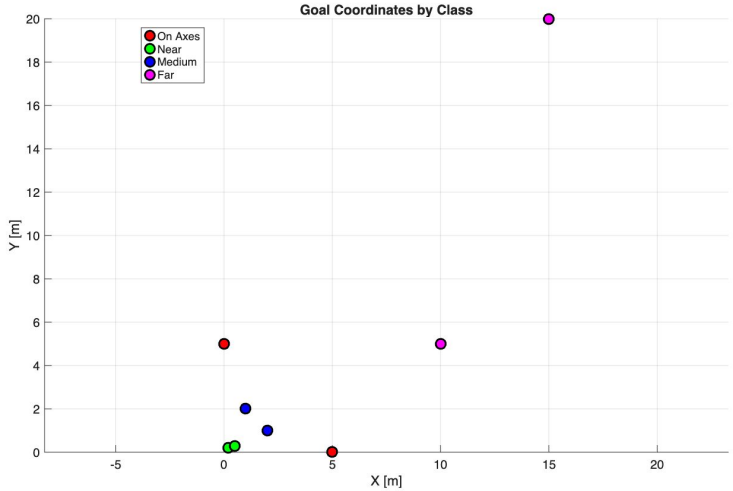
The 8 goals

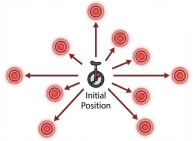
Regulation controllers were tuned using 8 target positions placed at varying distances from the initial position.

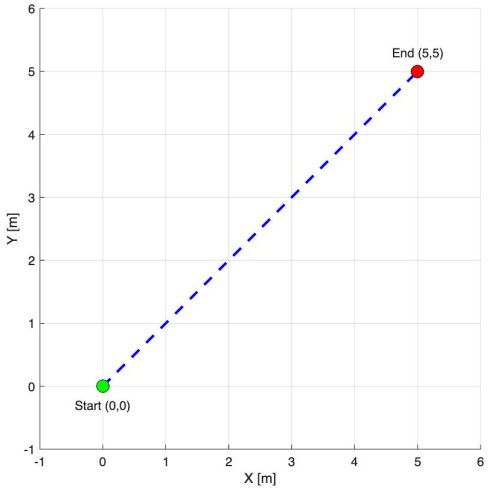
Cost function:

$$\rho(t) = \sqrt{(x(t) - x_d)^2 + (y(t) - y_d)^2}$$

$$J_{\text{parking}} = \frac{1}{\|[x_d, y_d]\|} \int_0^T \left(\rho(t)^2 + \alpha v(t)^2 + \beta \omega(t)^2\right) dt, \quad \alpha, \beta \in [0, 1]$$



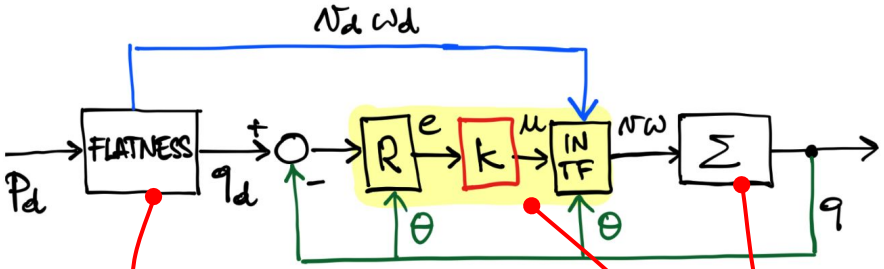




UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Tuning Trajectory Tracking Controllers

Trajectory tracking controller #1
State-error linearization control scheme



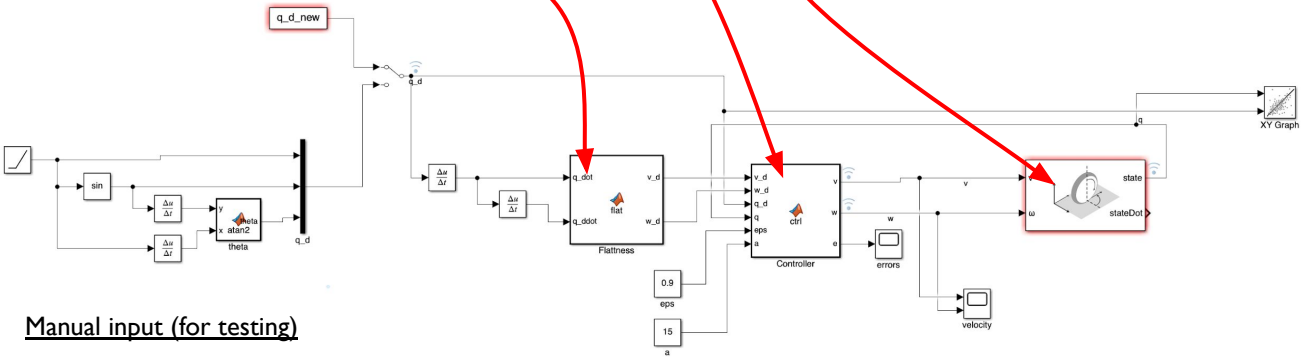
From theory:

$$\begin{cases} k1 = k3 = 2\xi a \\ k2 = \frac{a^2 - (\omega_d(t))^2}{v_d(t)} \end{cases} \quad \text{with} \quad \xi \in (0,1),\ a > 0$$

Chosen values:

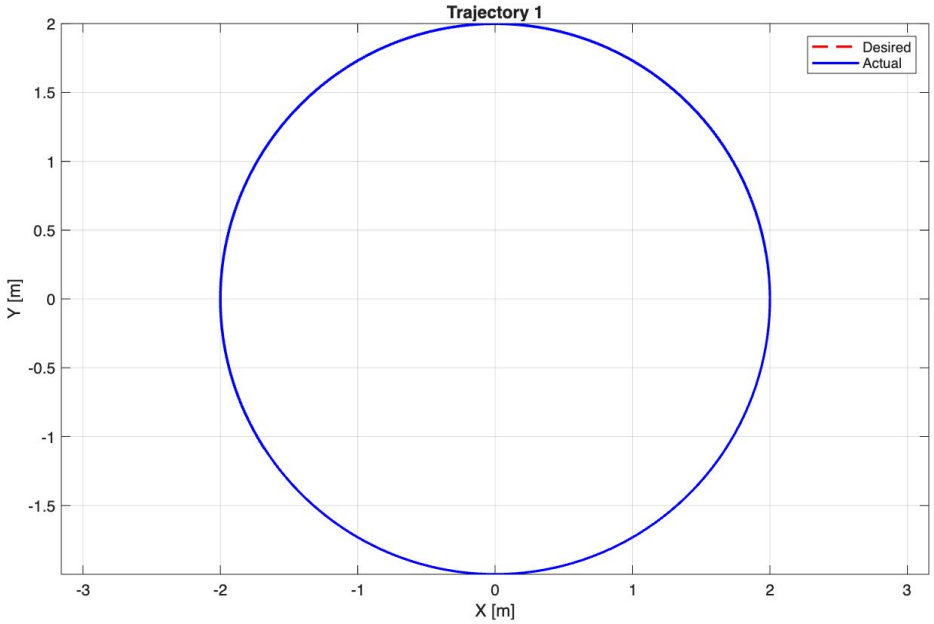
$$a = 20, \quad \xi = 0.87$$

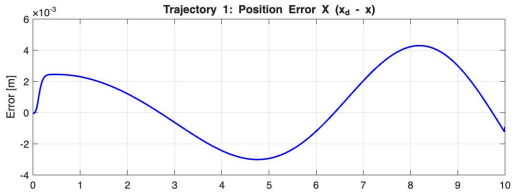
(natural frequency) (damping coeff)

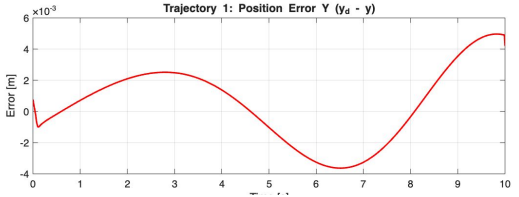


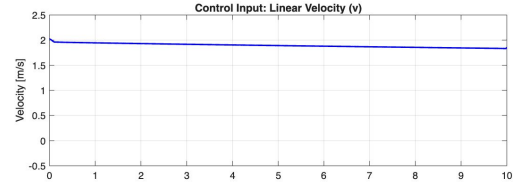
Manual input (for testing)

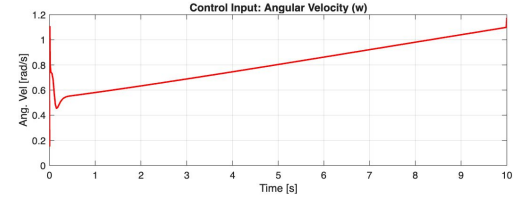
Trajectory 1



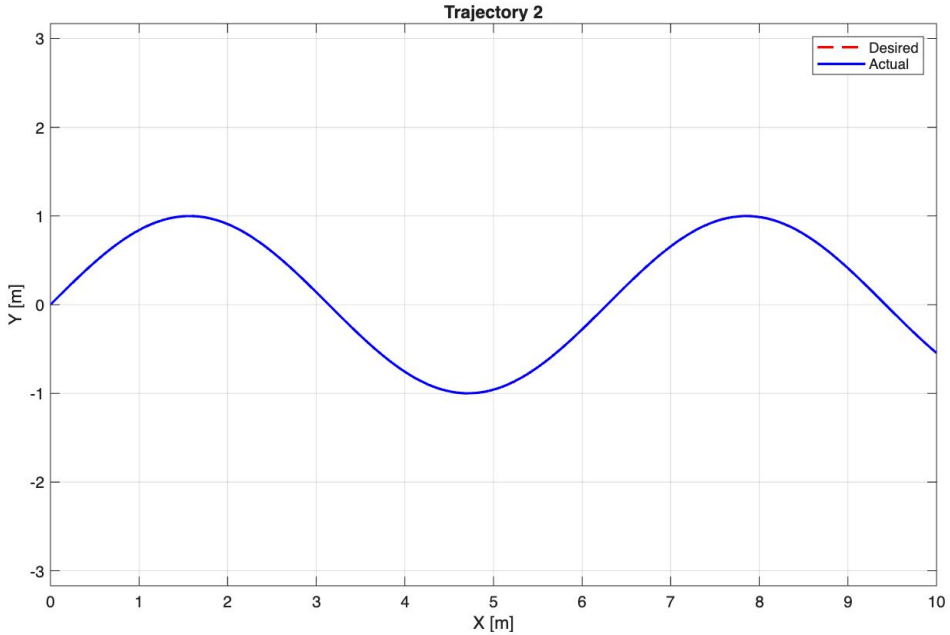


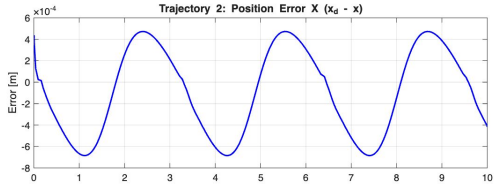


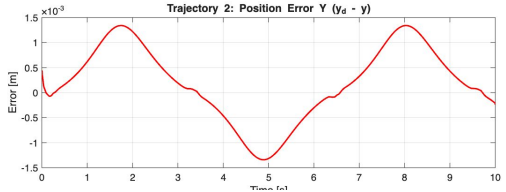


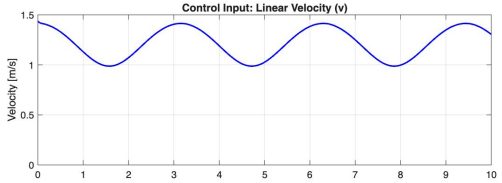


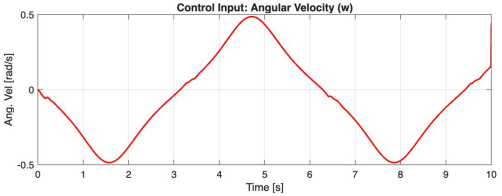
Trajectory 2



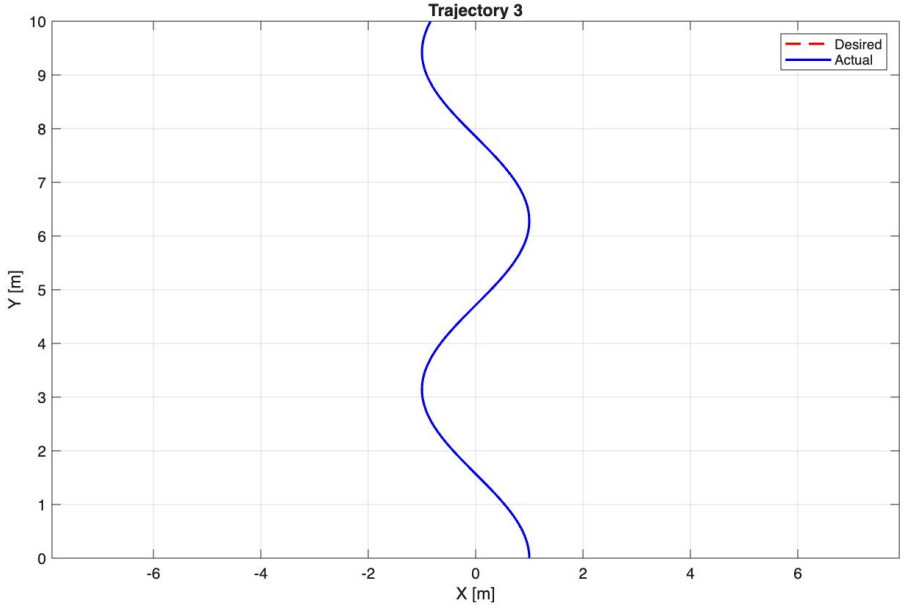


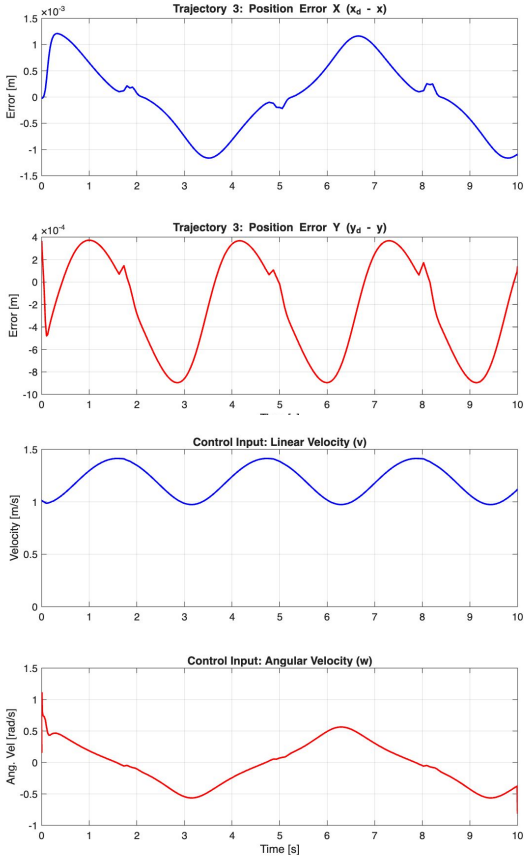




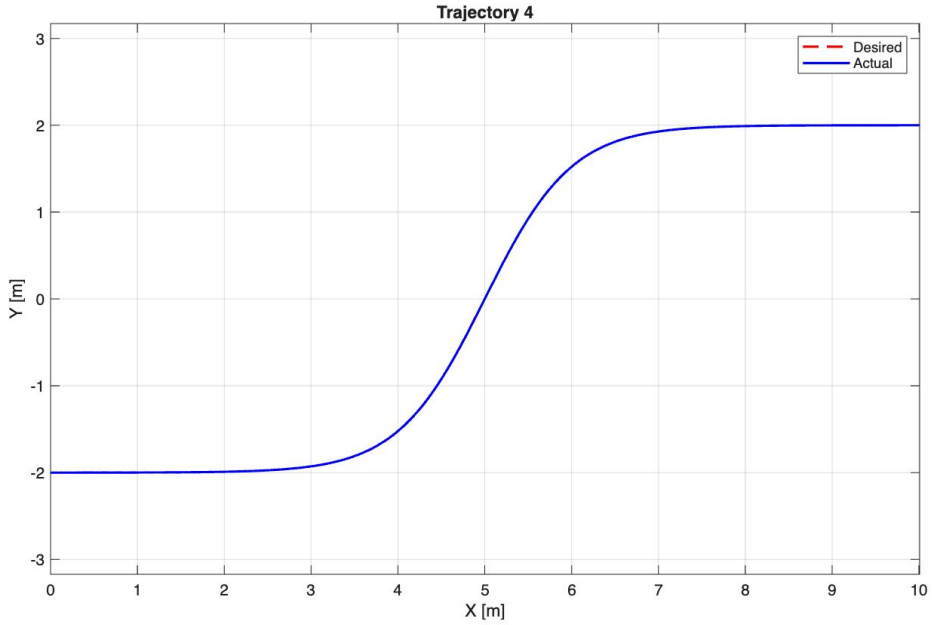


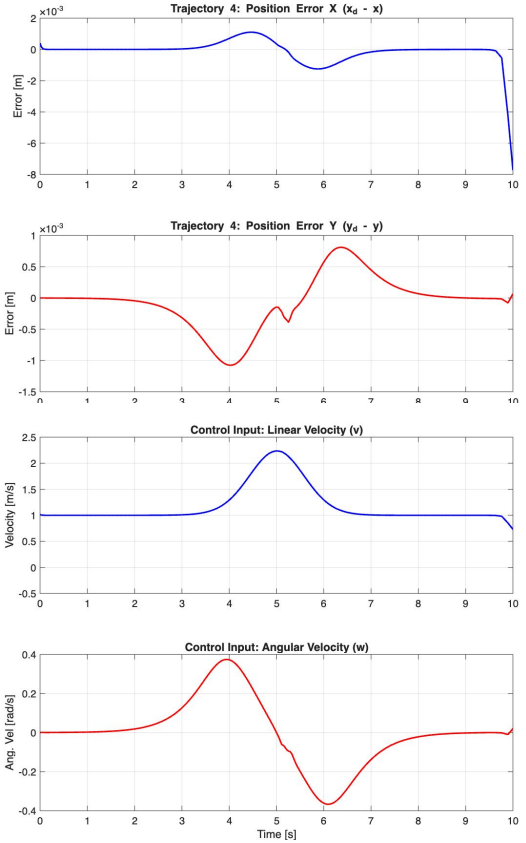
Trajectory 3





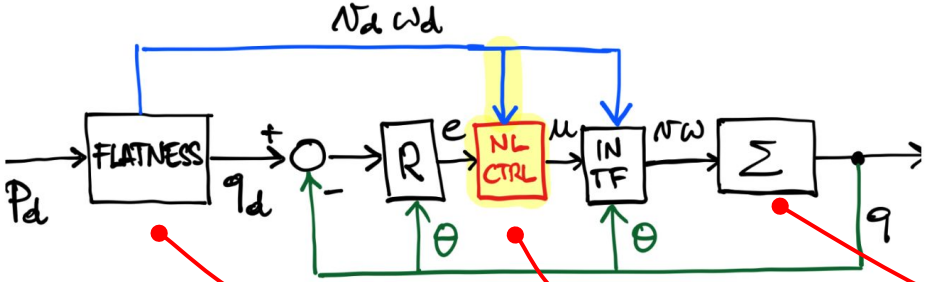
Trajectory 4





Trajectory tracking controller #2

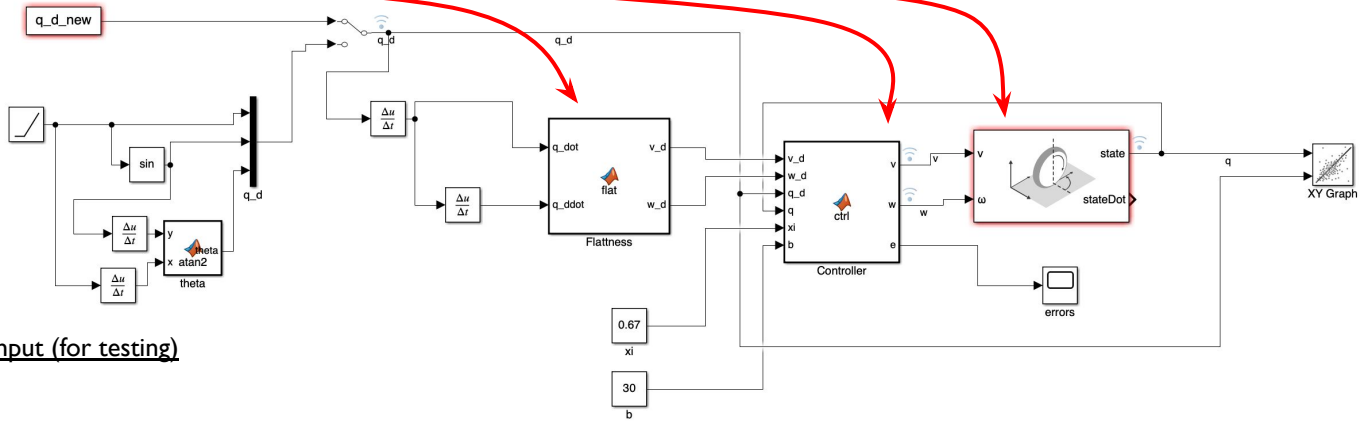
State-error nonlinear control scheme



From theory:

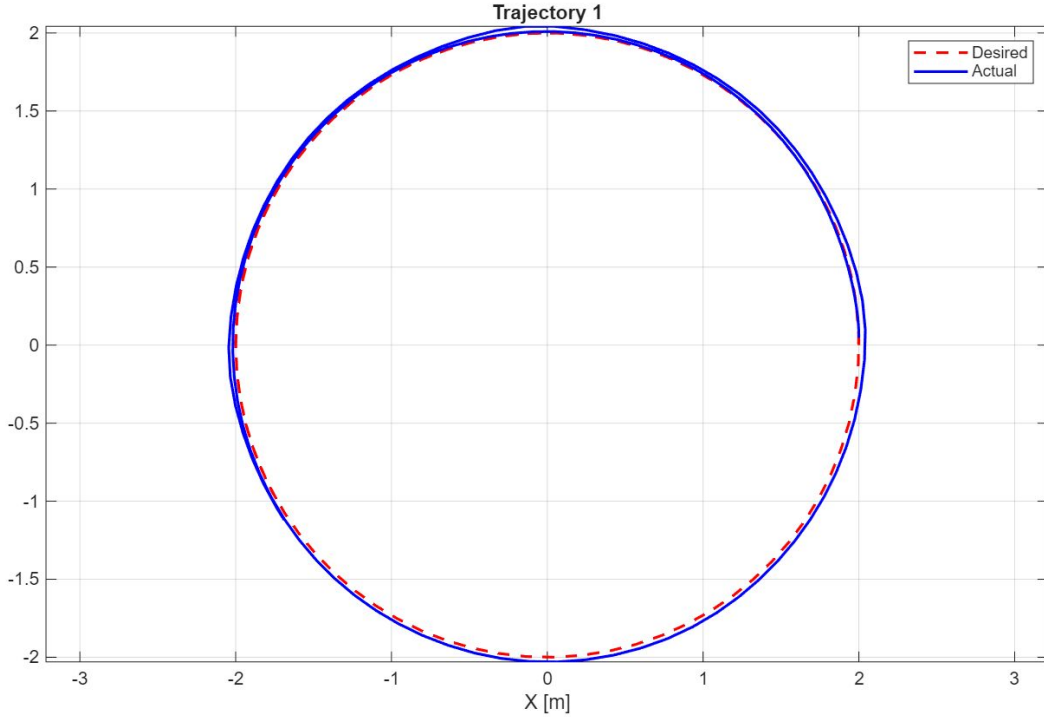
$$\begin{cases} \bar{k}_2 = b \\ k_1 = k_3 = 2\xi\sqrt{bv_d(t) + \omega_d^2(t)} \end{cases} \quad \text{with } b > 0,\ \xi \in (0,1)$$

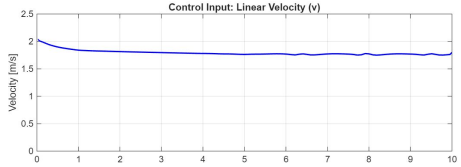
Chosen values: $b = 23.79 \quad \xi = 0.76$

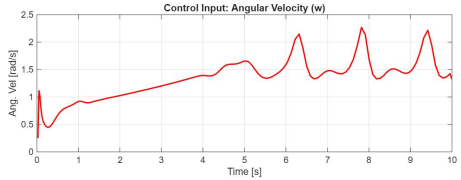


Manual input (for testing)

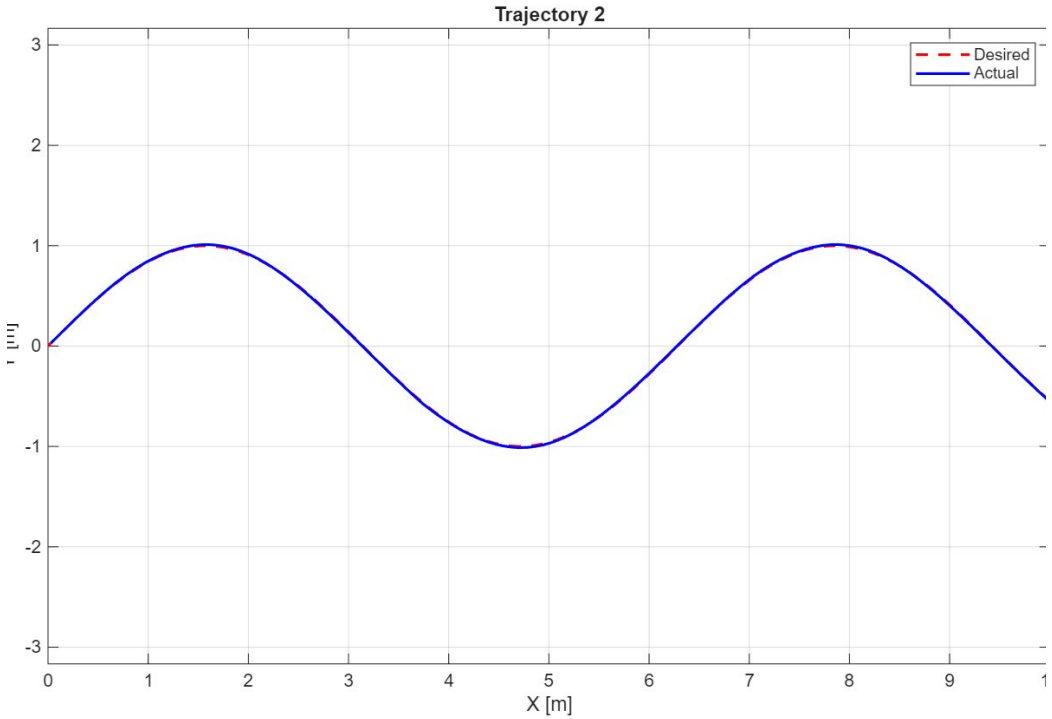
Trajectory 1

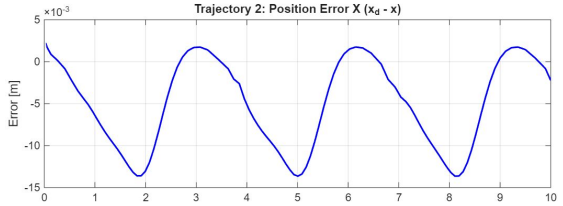


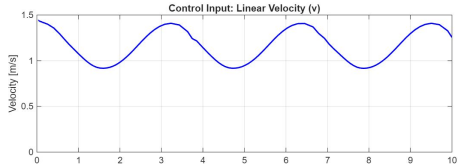


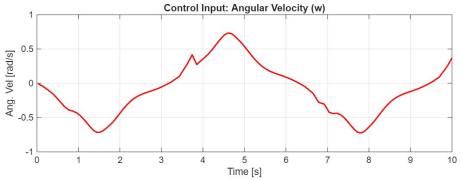


Trajectory 2

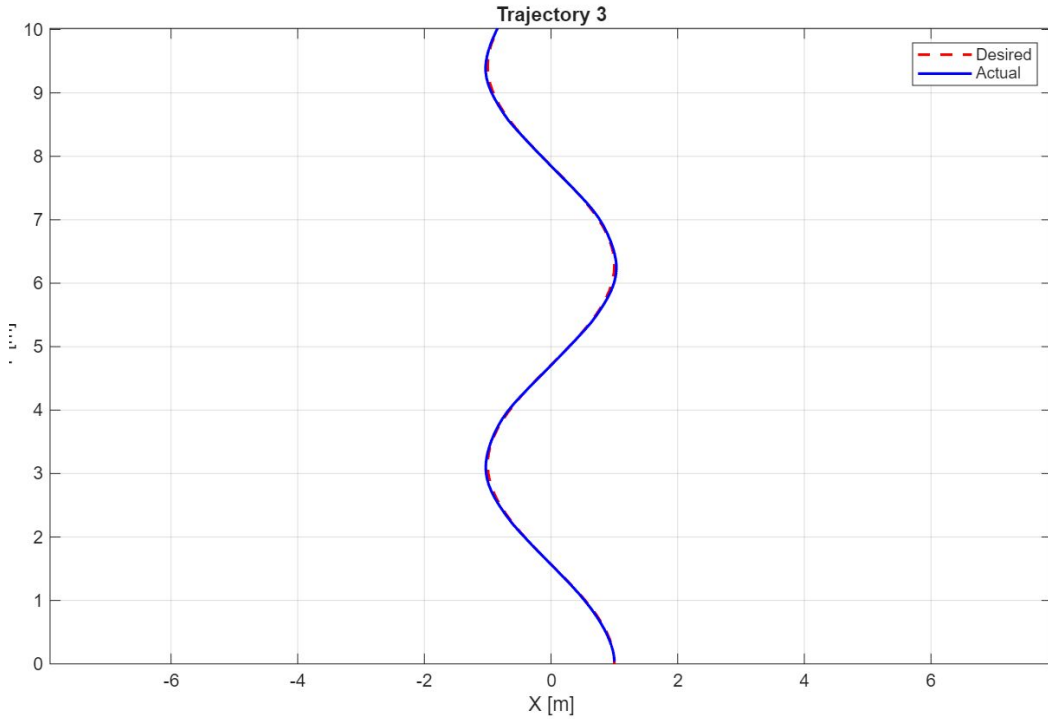


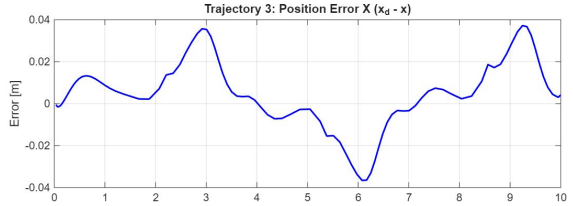


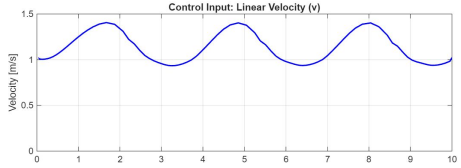


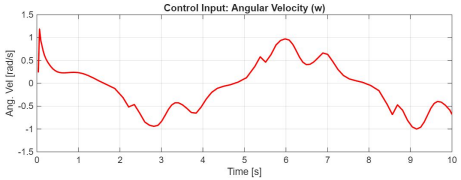


Trajectory 3

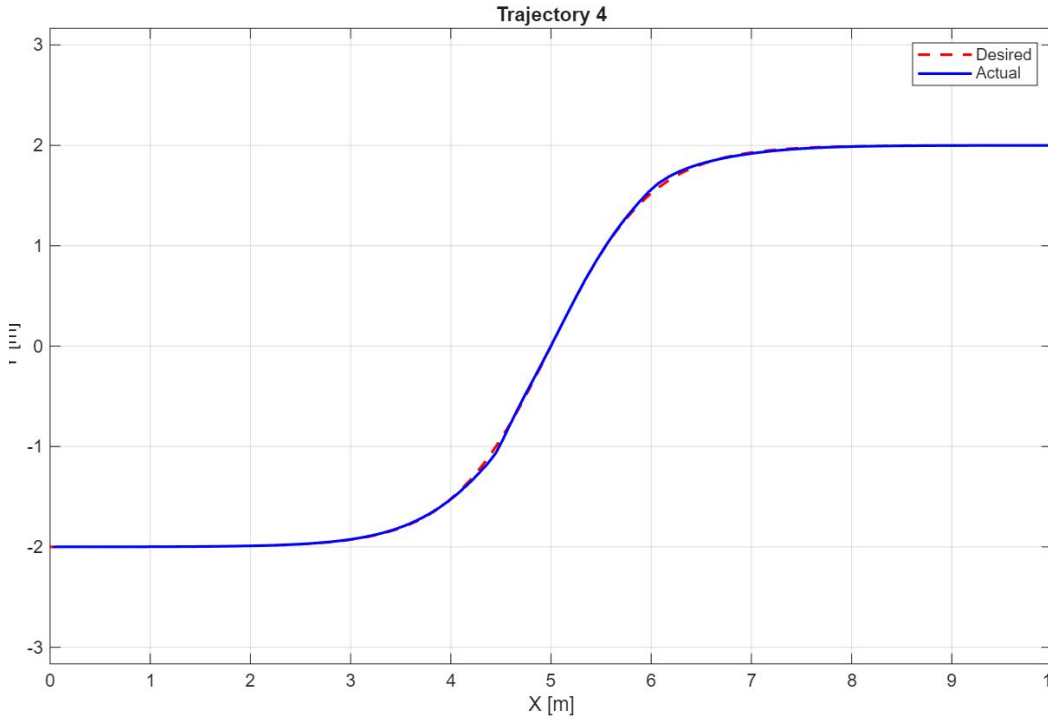


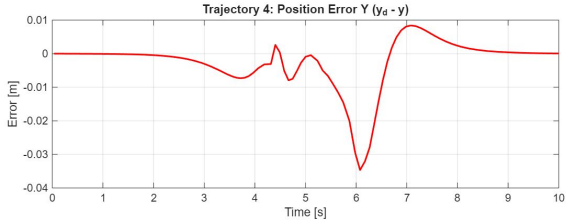


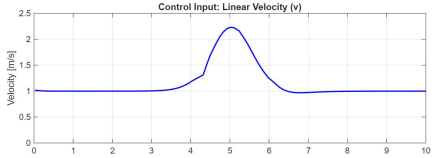


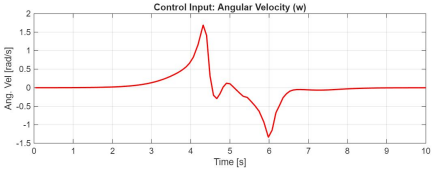


Trajectory 4









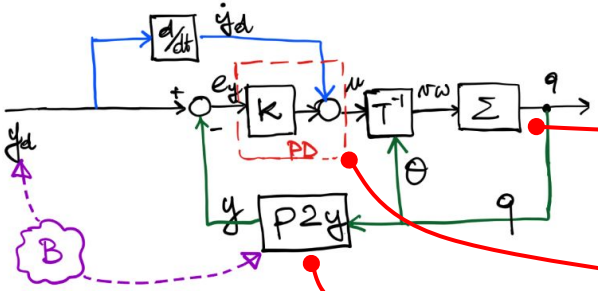
Trajectory tracking controller #3

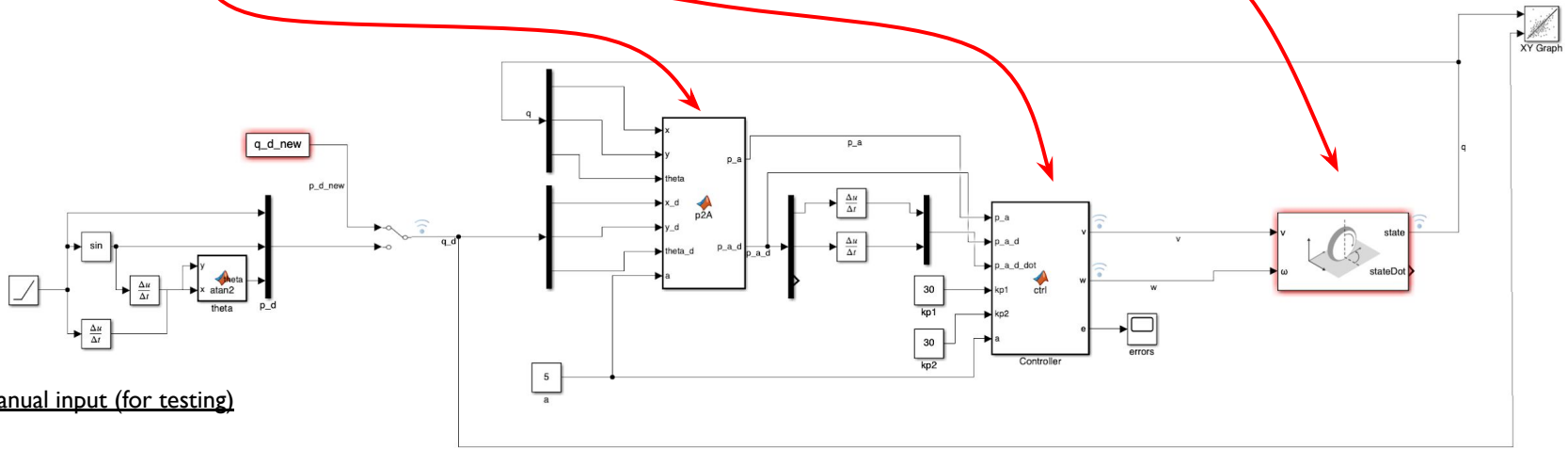
Output-error feedback control scheme

UNIVERSITÀ DEGLI STUDI DI PADOVA

From theory: $a \neq 0, \quad kp1 > 0, \quad kp2 > 0$

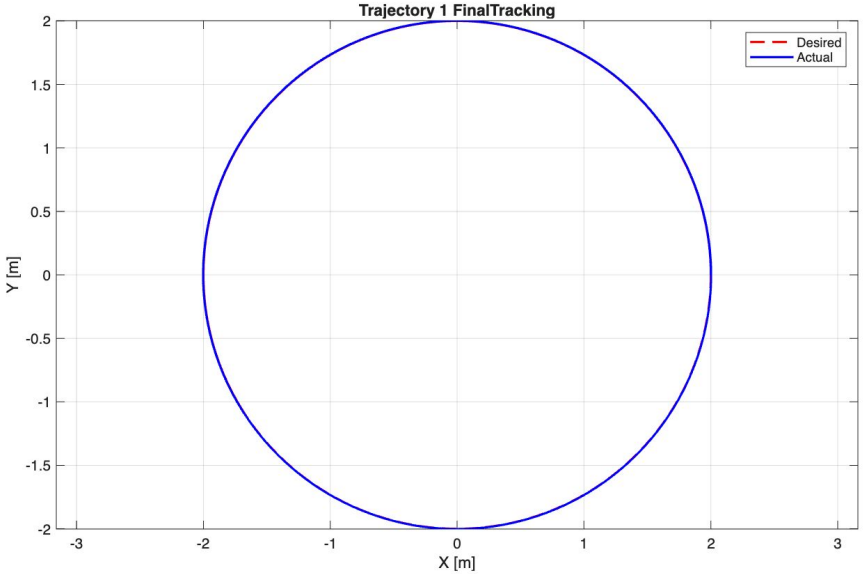
Chosen values: $a = 0.20, \quad kp1 = 11.50, \quad kp2 = 13.52$

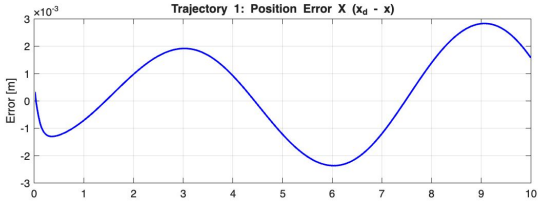


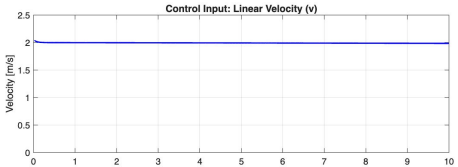


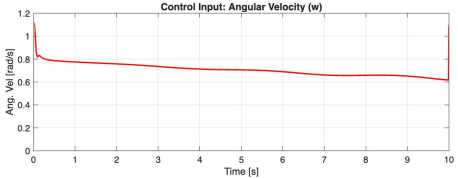
Manual input (for testing)

Trajectory 1

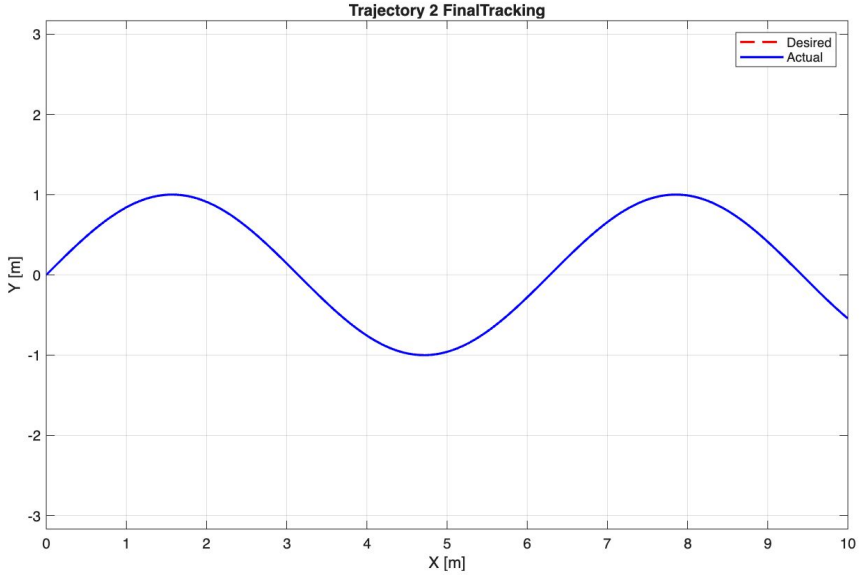


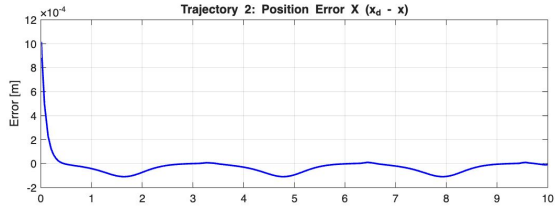


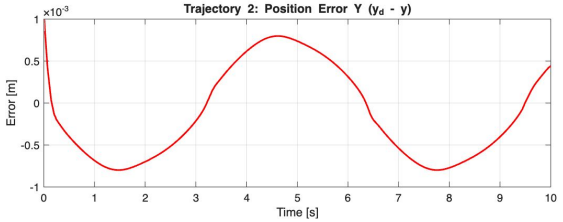


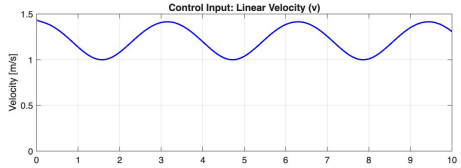


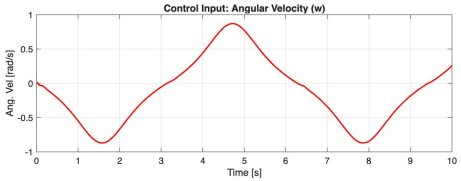
Trajectory 2



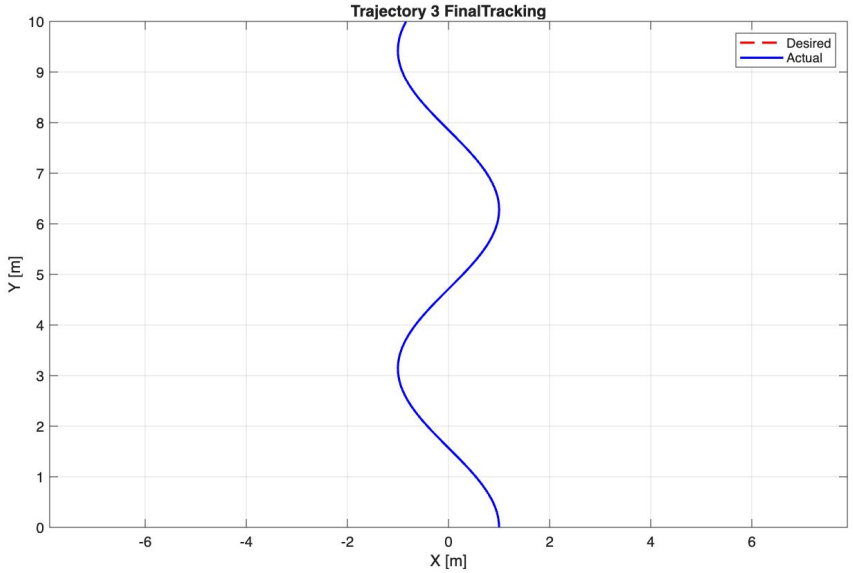


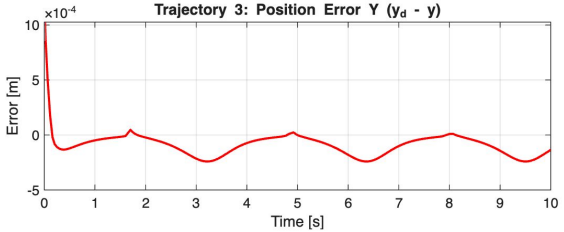


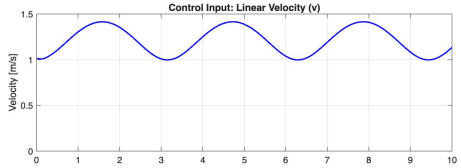


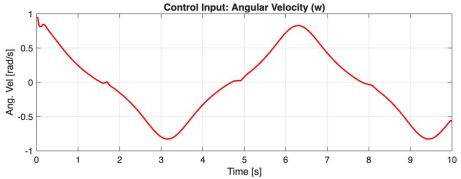


Trajectory 3

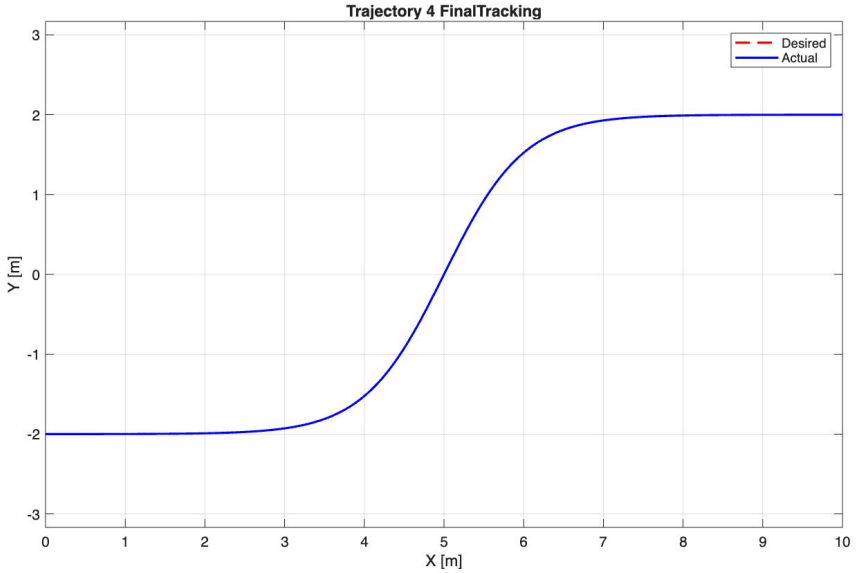


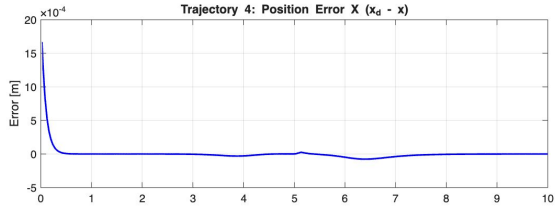


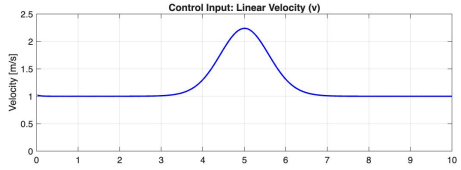


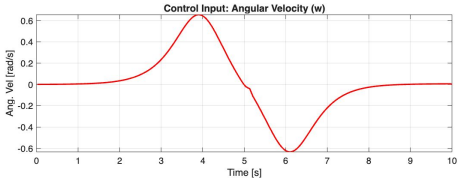


Trajectory 4

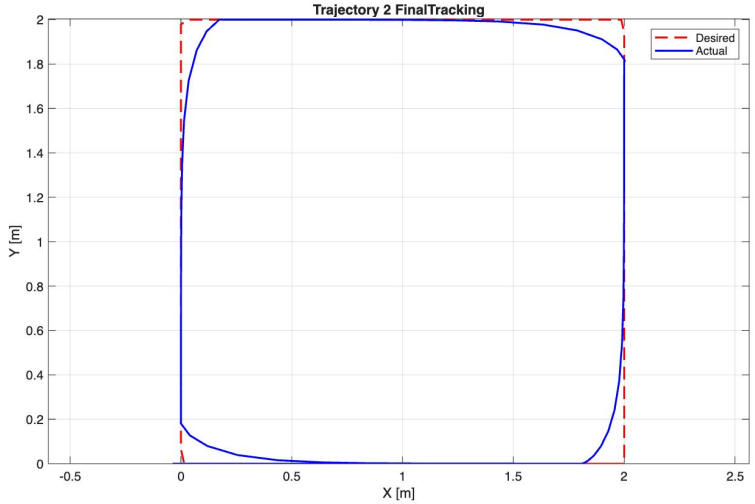




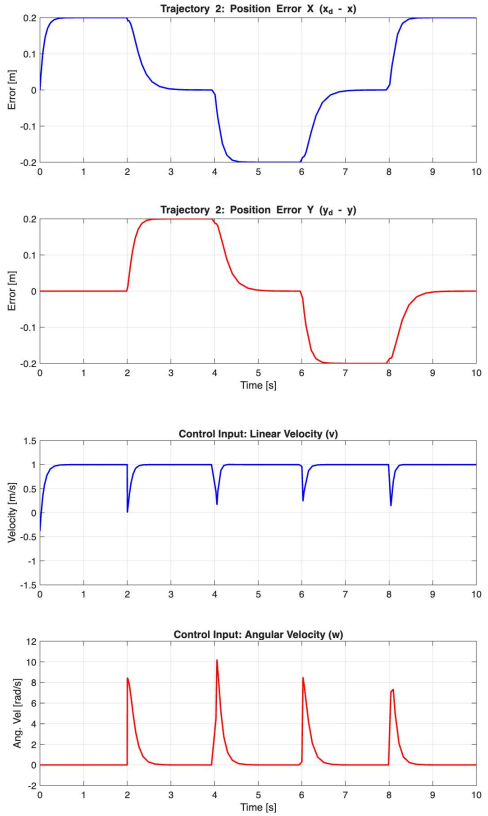




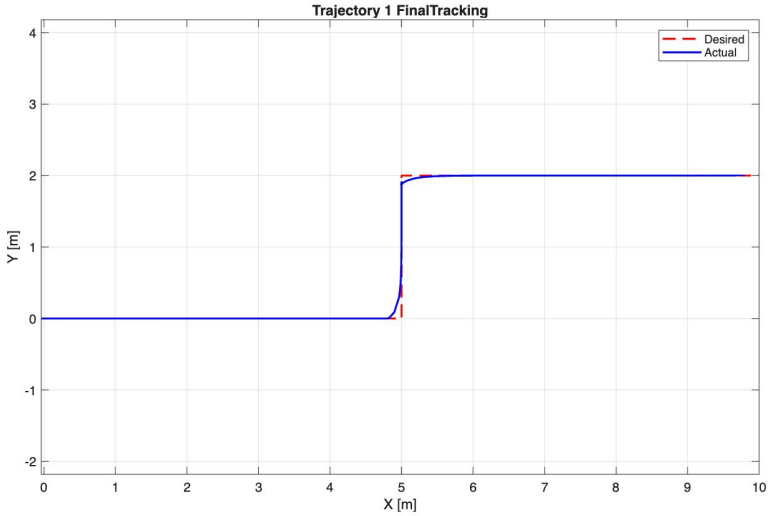
Trajectory 5



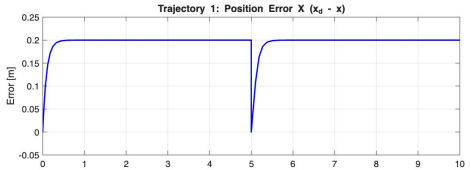
!

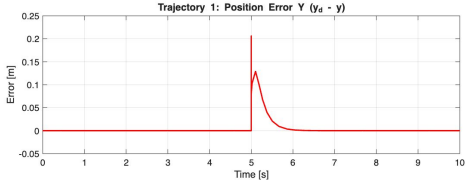


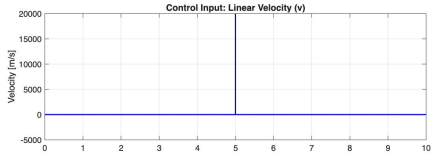
Trajectory 5

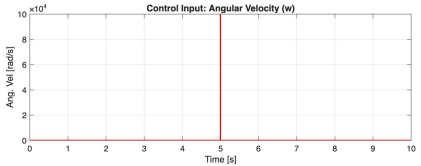


!

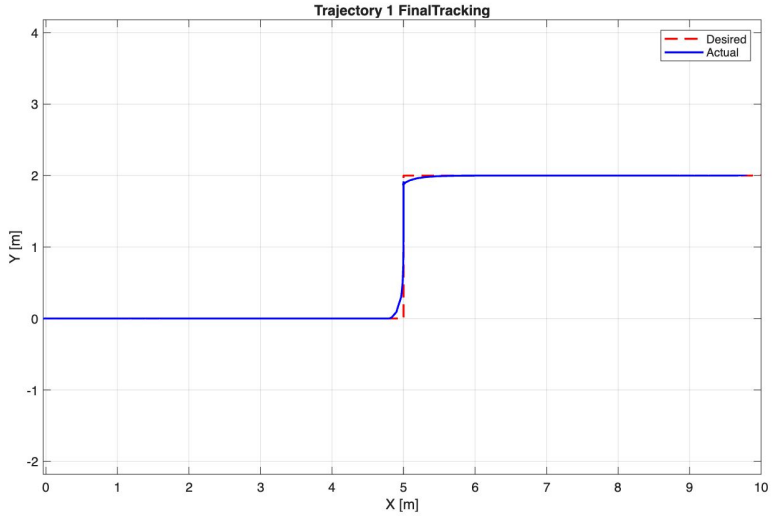


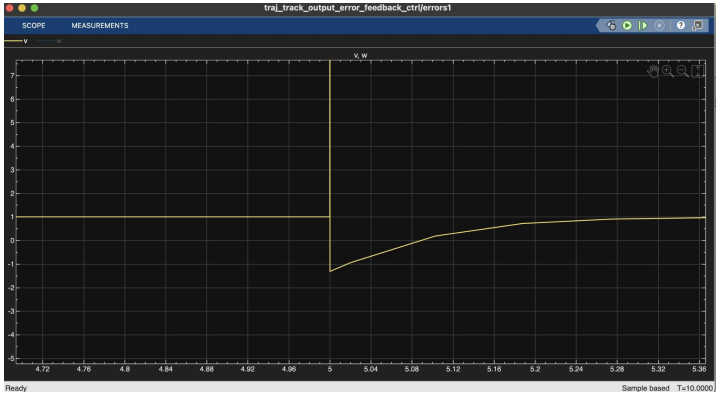


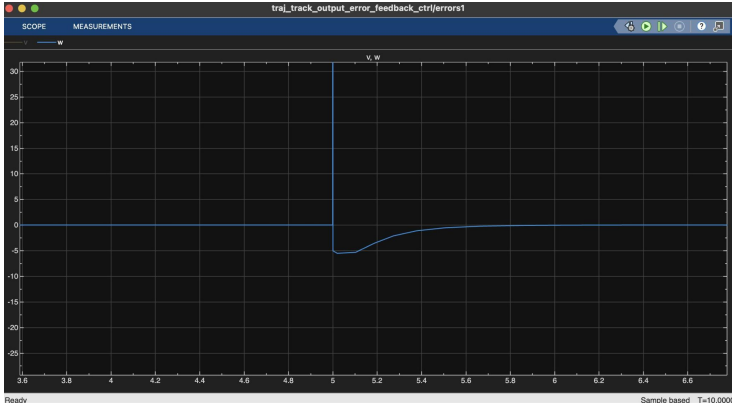




Trajectory 5: zoom







Trajectory tracking controllers comparison

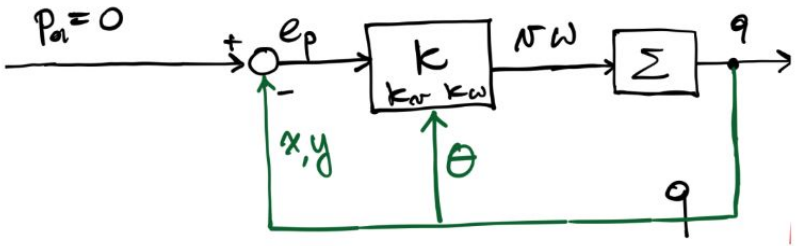
UNIVERSITÀ DEGLI STUDI DI PADOVA

Controller	State Error Linearized	State Error Non-Linear	Output Error Feedback
Strengths	Design simplicity; excellent for small deviations.	Handles large initial errors	Eliminates kinematic singularities of the robot.
Weaknesses	Performance drops if error is large; sensitive to phase jumps (±π).	Requires precise tuning (b,ξ); sensitive to phase jumps (±π).	The robot's center may oscillate while point a remains stable.
Average error values of the cost function	Lowest (~0.0016)	*Medium (~0.0151)*	*Medium-Low (~0.0099)*

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Tuning Cartesian and Posture Controllers

Cartesian Control

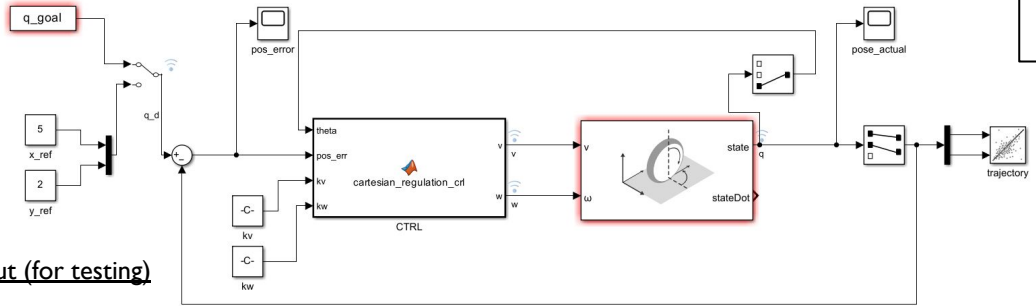


From theory:

$$\begin{cases} v = k_v e_p^T n = -k_v(x\cos\theta + y\sin\theta) \\ w = k_w\gamma = k_w(\text{atan2}(y,x) + \pi - \theta) \\ k_v, k_w > 0 \end{cases}$$

Chosen values:

$$\mathrm{K}_v = 2.79, K_w = 7.42$$

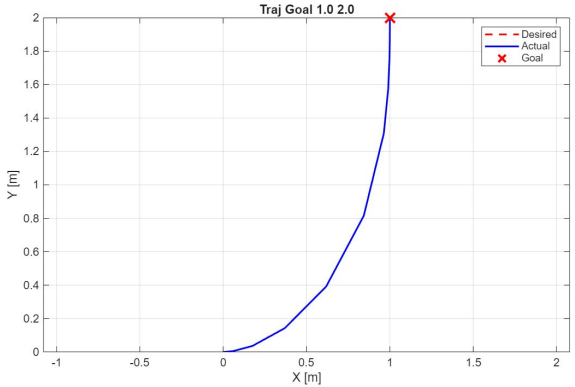


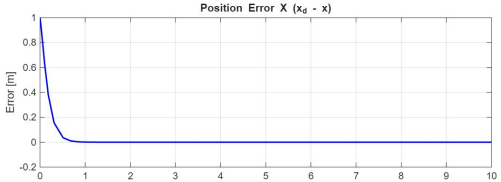
Manual input (for testing)

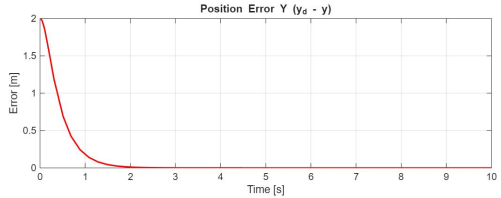
Tested Goals

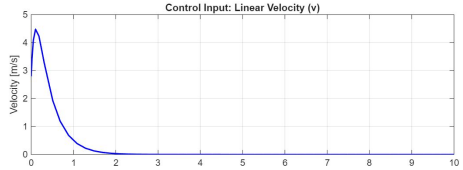
The first goal tested for cartesian controller was a near starting position:

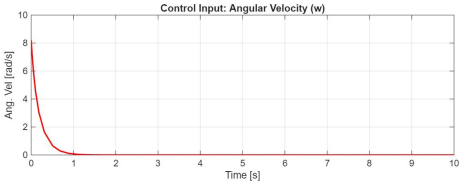
With the parametrization obtained from tuning the trajectory, position error and control input are the following.







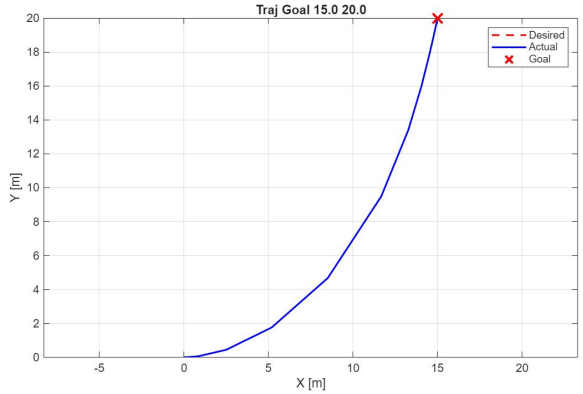


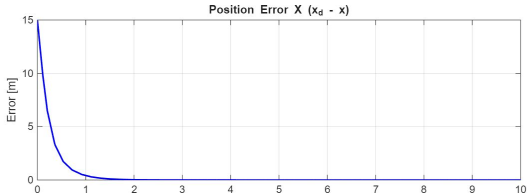


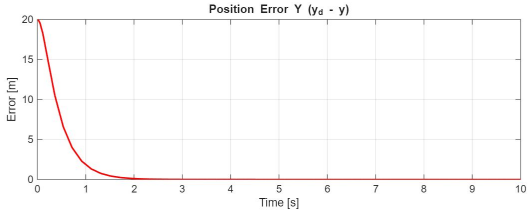
Tested Goals

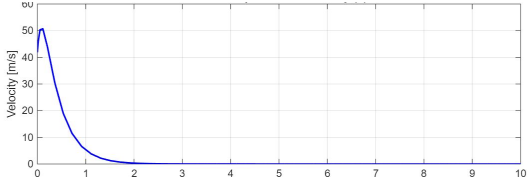
The last goal tested for cartesian controller was far from starting position.

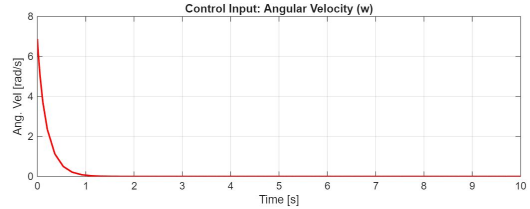
With the parametrization obtained from tuning the trajectory, position error and control input are the following.



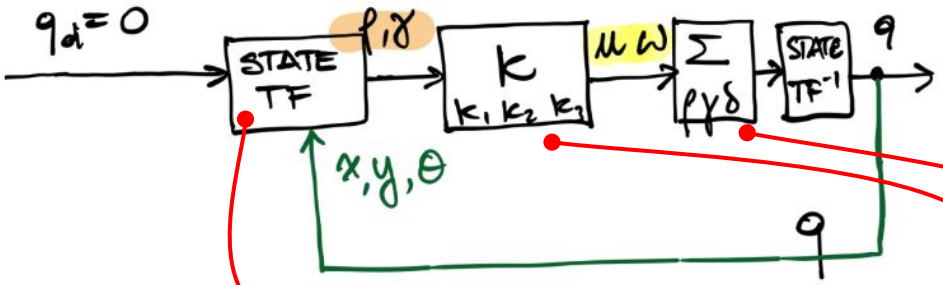








Posture Control

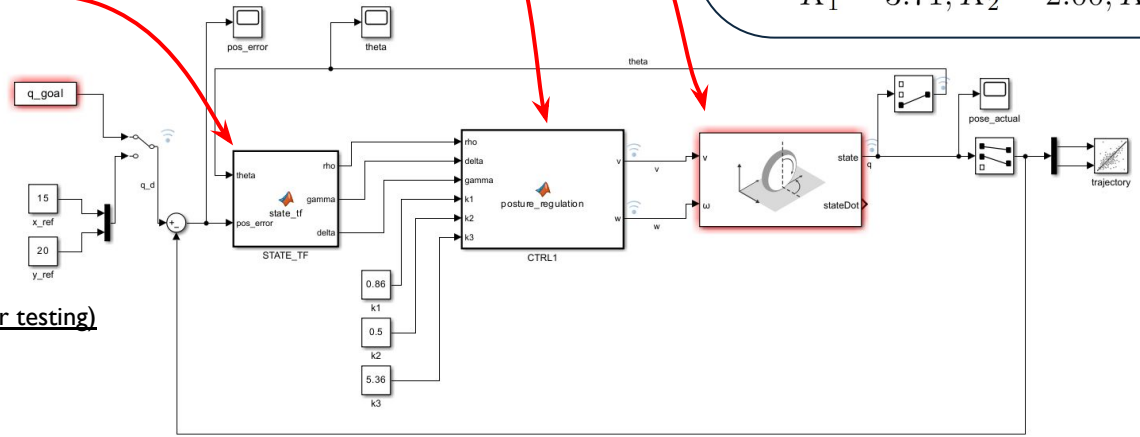


From theory:

$$\begin{cases} v = k_1 \rho \cos\gamma \\ w = k_2\gamma + k_1 \frac{\sin\gamma\cos\gamma}{\gamma}(\gamma + k_3\delta) \\ k_1, k_2, k_3 > 0 \end{cases}$$

Chosen values

$$K_1 = 3.71, K_2 = 2.00, K_3 = 6.00$$

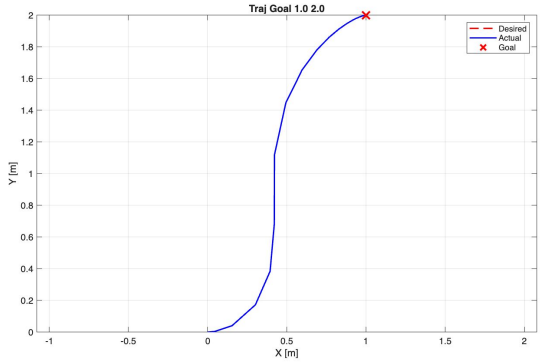


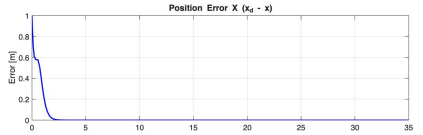
Manual input (for testing)

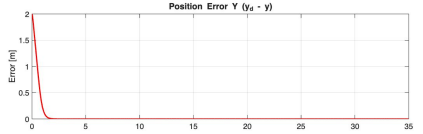
Tested Goals

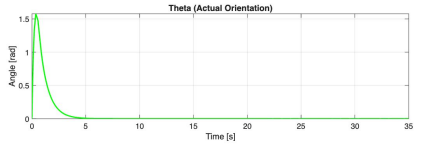
The first goal tested for posture cartesian controller was a near starting position.

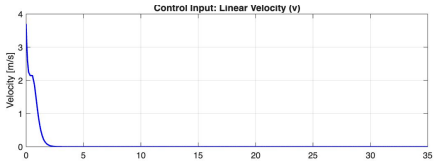
With the parametrization obtained from tuning the trajectory, position error and control input are the following.

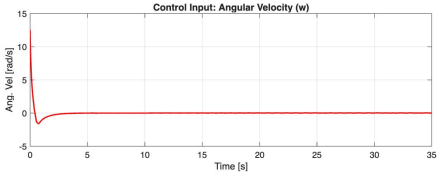








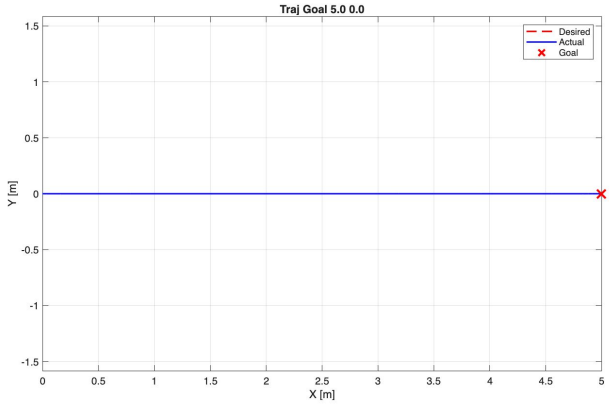


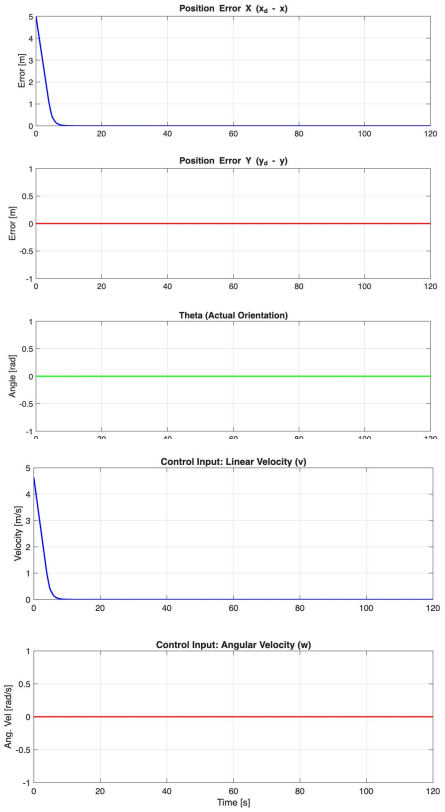


Tested Goals

The second goal tested for posture cartesian controller was along x axis.

With the parametrization obtained from tuning the trajectory, position error and control input are the following.





Posture regulation ~ Problems

The **forced orientation problem**:

- Since the orientation is forces to 0 we have as a consequence that the regulation accumulates a big error when the agent is positioned to the right with respect to the goal

The **Vertical position of goal**:

- When the goal is instead vertically aligned with the starting position, the posture controller cannot reach the goal.

Posture regulation ~ Problems

Starting from initial position, setting angular error to **pi/2** (initial position oriented with angle 0 and vertical goal) we obtain the following:

$$\begin{cases} v = k_1 \rho \cos\gamma \\ w = k_2\gamma + k_1 \frac{\sin\gamma\cos\gamma}{\gamma}(\gamma + k_3\delta) \\ k_1, k_2, k_3 > 0 \end{cases} \longrightarrow \begin{cases} \cos\gamma = 0 \\ v = 0 \\ w = k_2\frac{\pi}{2} \\ k_2 > 0 \end{cases}$$

Result: constant rotation around initial position.

Regulation controllers comparison

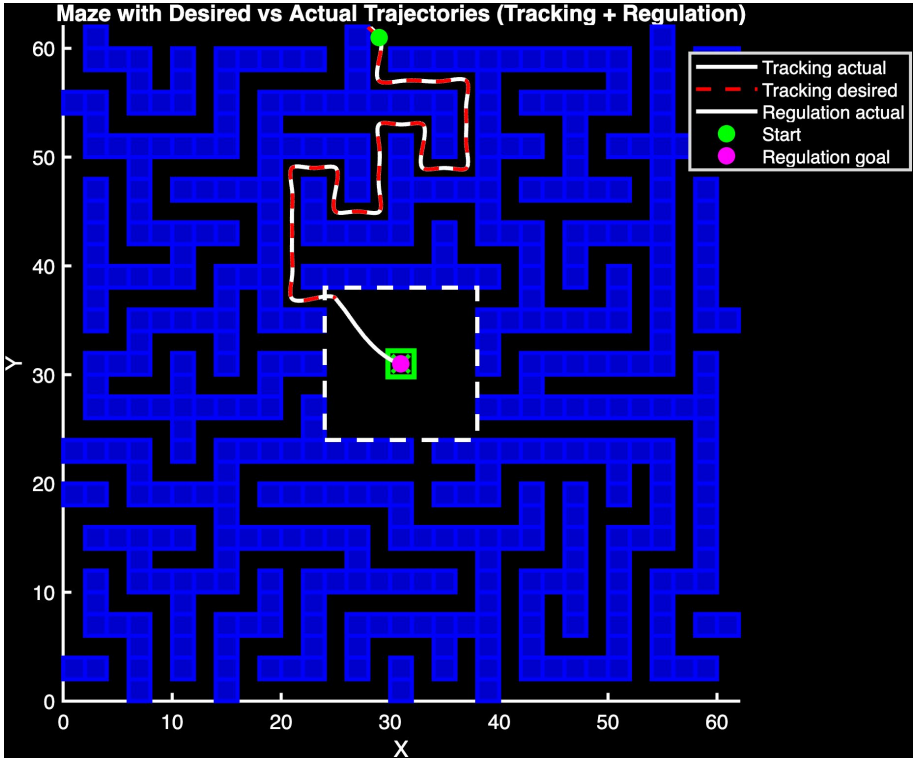
UNIVERSITÀ DEGLI STUDI DI PADOVA

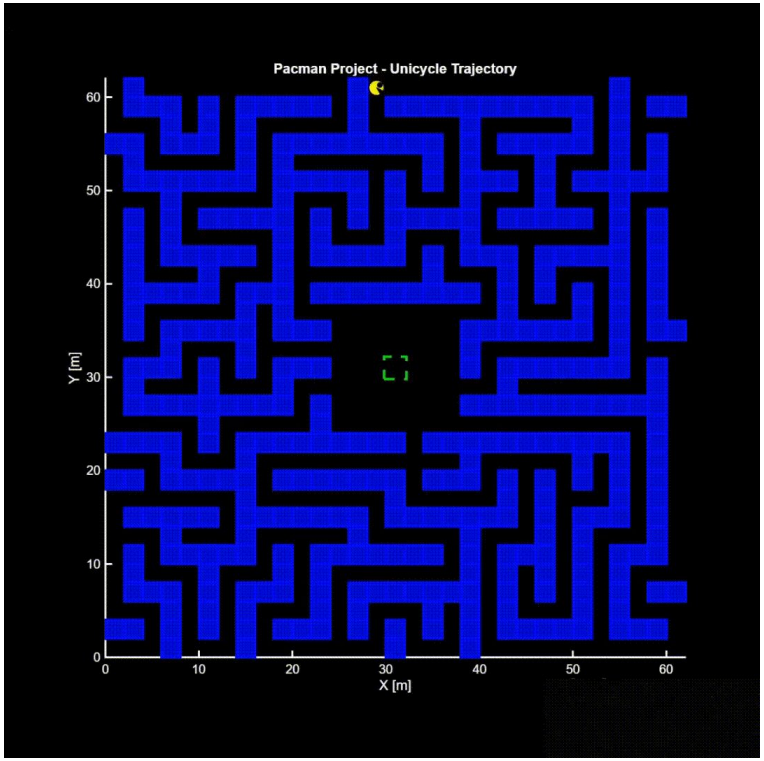
Controller	**Cartesian regulation**	**Posture regulation**
Strengths	Design simplicity; Faster control	More precise; Consider orientation
Weaknesses	Too simple depending on the goal of the controller	Requires more tuning (has 3 gains); sensitive to angle error value(γ)
Average error values of the cost function	*Lowest (~2.6603)*	*Biggest (~3.3019)*

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Testing controllers in the Maze

PACMAN sim with the controllers tuned

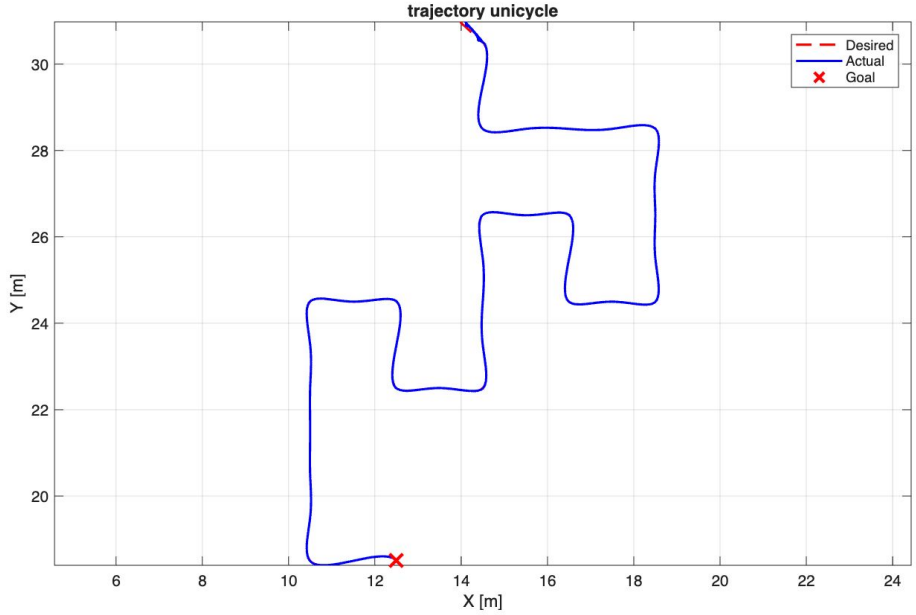


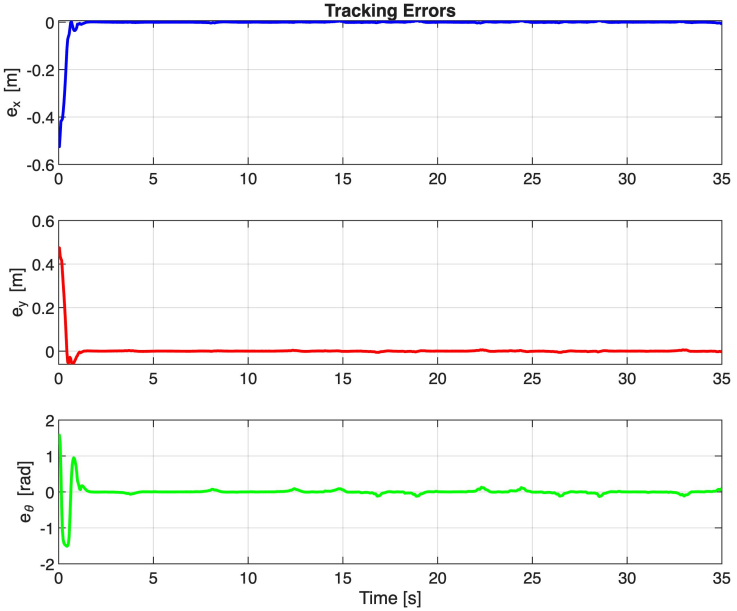


PACMAN simulation

with Trajectory Tracking Linearized Controller

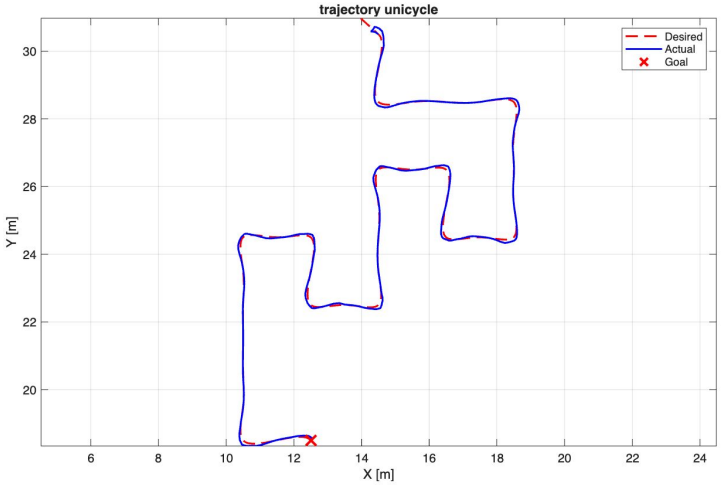
Università degli Studi di Padova

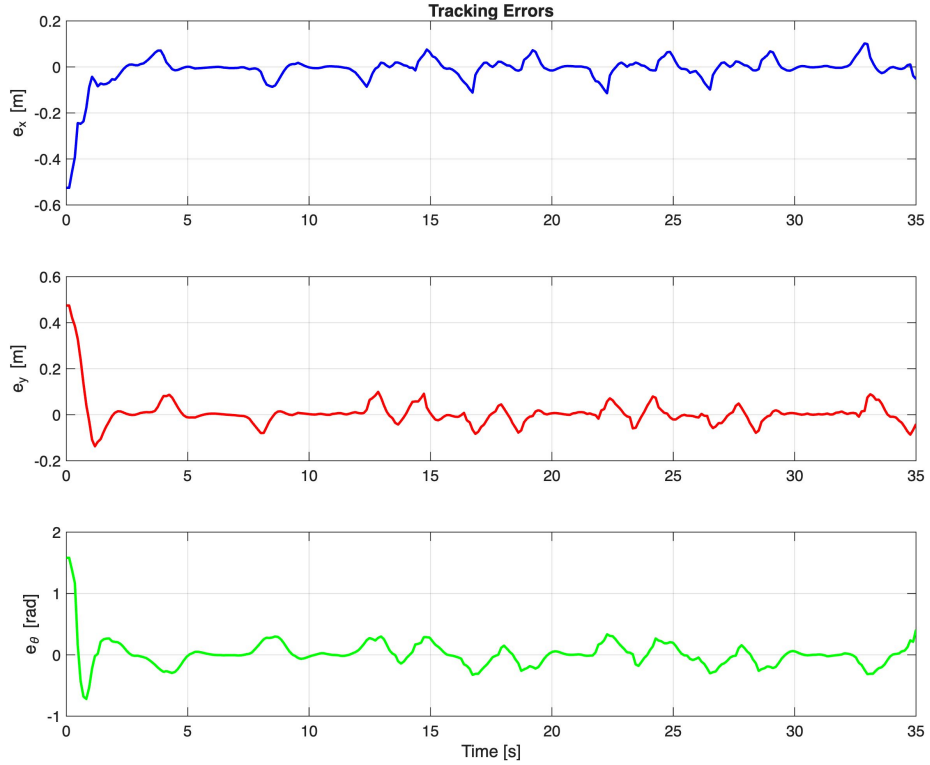




PACMAN simulation

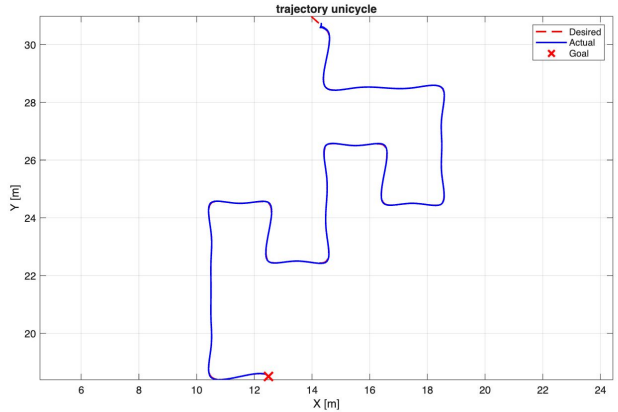
with Trajectory Tracking Nonlinear Controller

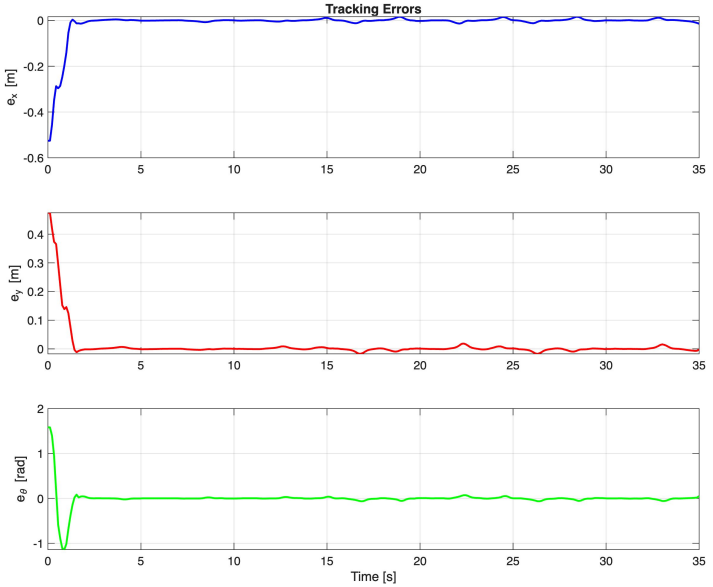




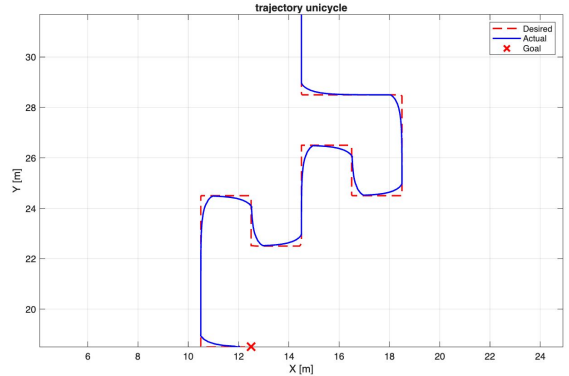
PACMAN simulation

with Trajectory Tracking Output Error Feedback Controller





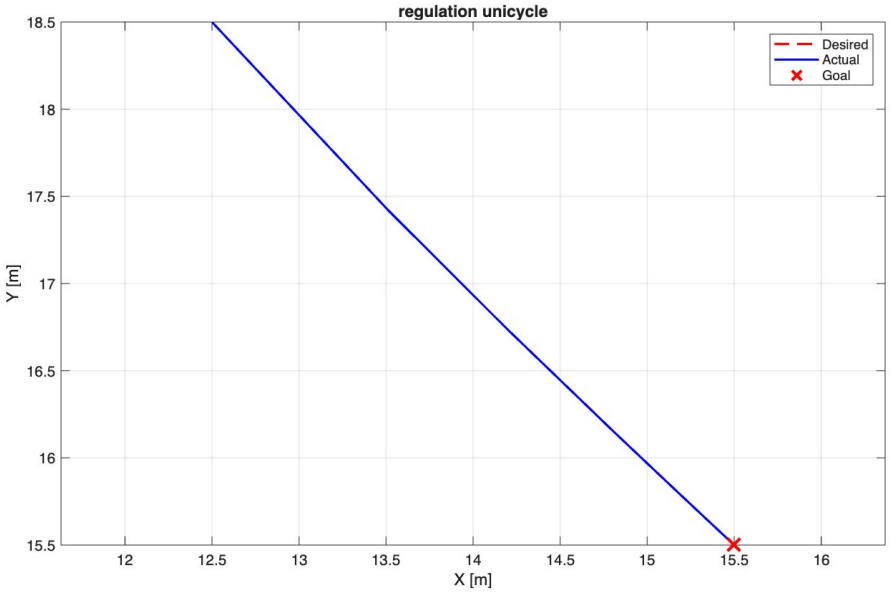
Trajectory with corners

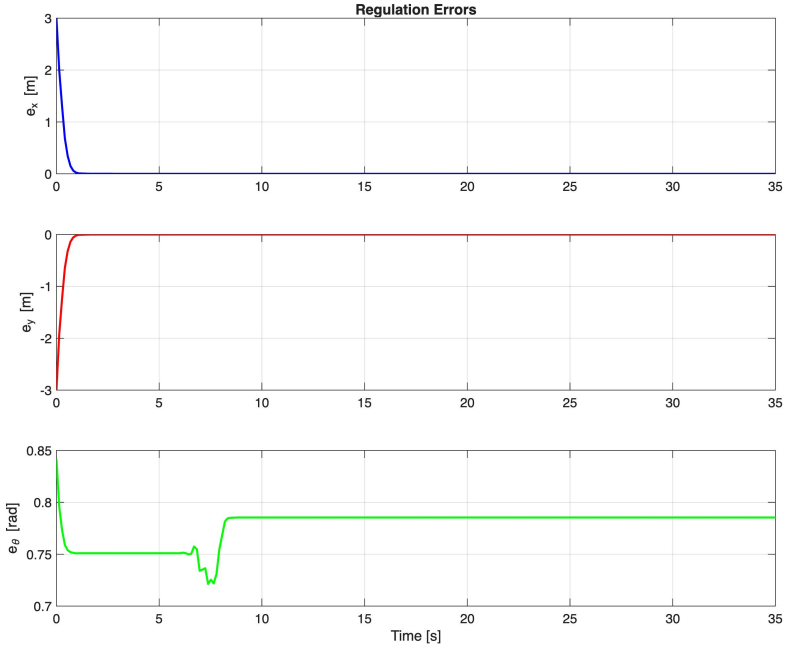


PACMAN simulation

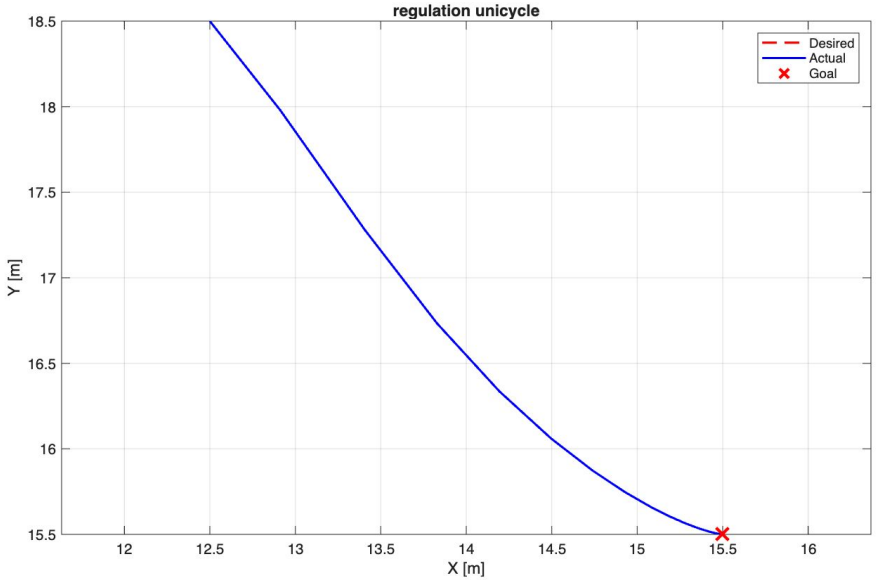
with Cartesian Regulation Controller

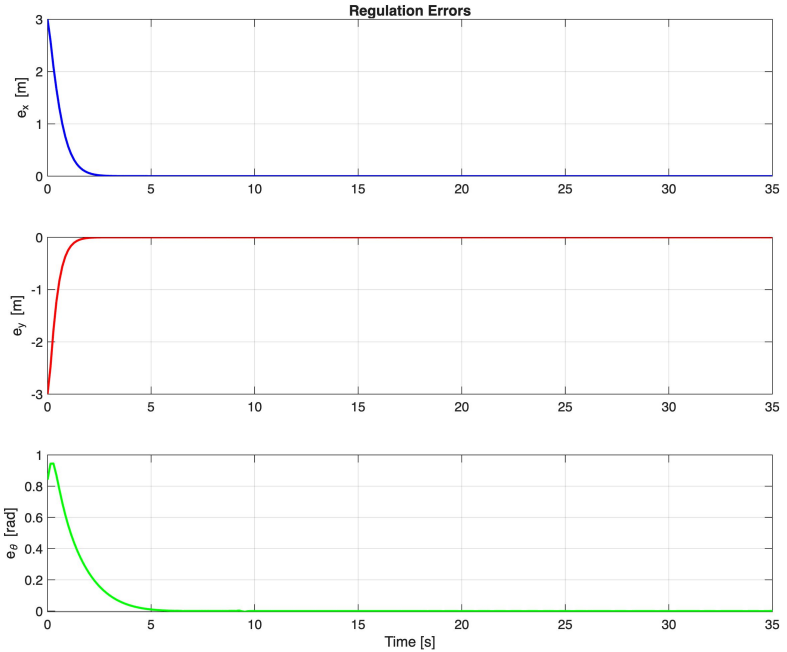
Università degli Studi di Padova





PACMAN simulation
with Posture Regulation Controller





Conclusions

Controller	Performance	Highlights
State Error Linearized	Excellent steady-state performance	Least oscillation once the trajectory is locked
State Error Nonlinear	Persistent oscillations, it needs an exhaustive tuning	It maintains tracking throughout the entire path
Output Error Feedback	Handles both curved trajectories and segments with 90° corners	Errors return rapidly to zero even after sudden changes
Cartesian regulation	Position errors go to zero	Reaches the target (x,y)
Posture regulation	All three errors converge asymptotically to zero	Essential if the goal requires also a specific orientation pose

Thank you for the attention