Università degli Studi di Napoli “Federico II”

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL’INFORMAZIONE

FIELD AND SERVICE ROBOTICS (FSR)
HOMEWORK n. 2

Prof. Eng. Fabio Ruggiero

Students:
Alberto Guglielmo Matr. P38000261

ACADEMIC YEAR 2024 / 2025

INDEX

CAPITOLO 1. PATH PLANNING ALGORITHM

1. Implement via software the path planning algorithm for a unicycle based on a cubic Cartesian polynomial. Plan a path leading the robot from the configuration $q_i = [x_i \quad y_i \quad \theta_i]^T = [0 \quad 0 \quad 0]^T$, to a random configuration $q_f = [x_f \quad y_f \quad \theta_f]^T$ generated automatically by the code such that $\|q_f - q_i\| = 1$. Then, determine a timing law over the path to satisfy the following velocity bounds $|v(t)| \leq 0.5$ m/s and $|\omega(t)| \leq 2$ rad/s. [*Hint: For the final configuration, use the command rand(1,3): save the result in a vector and divide it by its norm.*]

The chosen time law s(t) is characterized by a cubic polynomial whose trajectory goes from $s_i = 0$ to $s_f = 1$, with t_i = 0 and t_f = 1 seconds (initially assumed equal to 1, but will be scaled).

Given s(t) as a cubic function of the type

$$s(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0$$

$$\begin{cases} s(0) = 0 \\ s(1) = 1 \\ \dot{s}(0) = 0 \\ \dot{s}(1) = 0 \end{cases}$$

$$s(t) = -2t^3 + 3t^2$$

q_f was randomly extracted keeping $||q_f - q_i||$=1

```
qf_rand = rand(1,3);          % random 3D vector
qf_rand = qf_rand / norm(qf_rand);  % normalize
```

$$x(s) = s^3 x_f - (s-1)^3 x_i + \alpha_x s^2 (s-1) + \beta_x s (s-1)^2$$
$$y(s) = s^3 y_f - (s-1)^3 y_i + \alpha_y s^2 (s-1) + \beta_y s (s-1)^2$$

$$\alpha_x = k \cos\theta_f - 3x_f$$
$$\alpha_y = k \sin\theta_f - 3y_f$$
$$\beta_x = k \cos\theta_i + 3x_i$$
$$\beta_y = k \sin\theta_i + 3y_i$$

choose k=1;

The values $x, \dot{x}, \ddot{x}, y, \dot{y}, \ddot{y}, s, \dot{s}, \ddot{s}, \tilde{w}, \tilde{v}, w, v, \theta$ were computed using the formulas which, for the sake of simplicity, are not reported in this document.

Once the results were obtained, the time law was scaled to ensure |v| ≤ 0.5 m/s and |w|≤ 2 rad/s.

The parameters of the function s(t) are modified according to the formula:

$$scaling\ value = \max\left(\frac{\max(|v(t)|)}{v\_max}\right), \max\left(\frac{\max(|w(t)|)}{w\_max}\right));$$

$$a_2 = \frac{3}{T_{scaled}^{\ 2}} \quad a_3 = -\frac{2}{T_{scaled}^{\ 3}}$$

When the maximum allowed velocity is exceeded, the trajectory is time-scaled.

An analysis was carried out to evaluate how much the actual velocities exceeded the maximum limit, and the excess was expressed as a percentage.

Among the two analyzed cases, the worst-case scenario (i.e., the one with the highest percentage of

violation) was selected.
Based on this, the new required travel time was computed, considering that increasing the travel time leads to a inversely proportional reduction of the velocity.
Obviously, after scaling the time law, the trajectory does not change, only the speed at which it is travelled

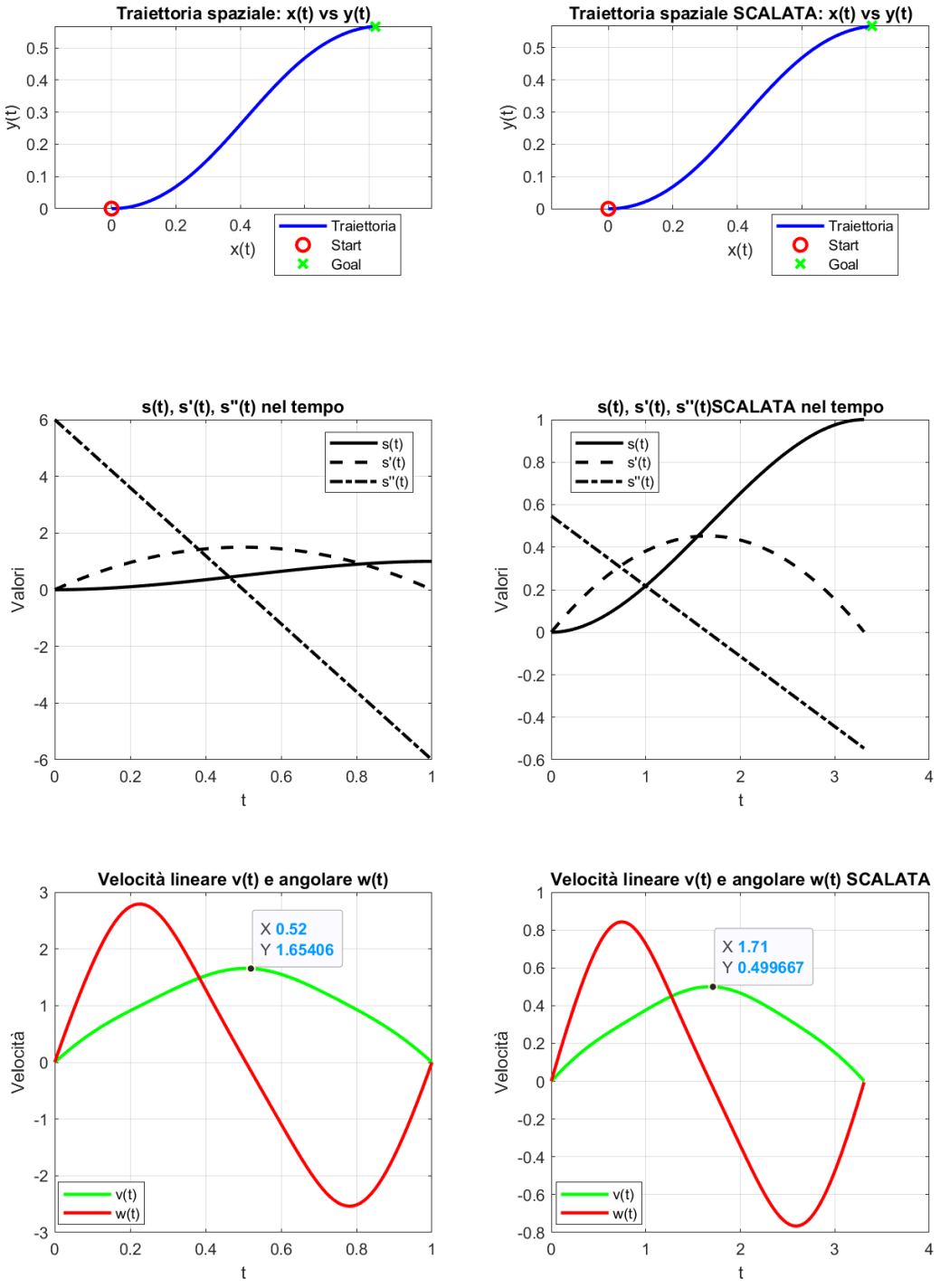


Fig. 1. path planning algorithm for unicycle, scaled and unscaled with final pose qf=[0.8190, 0.5670, 0.0875]

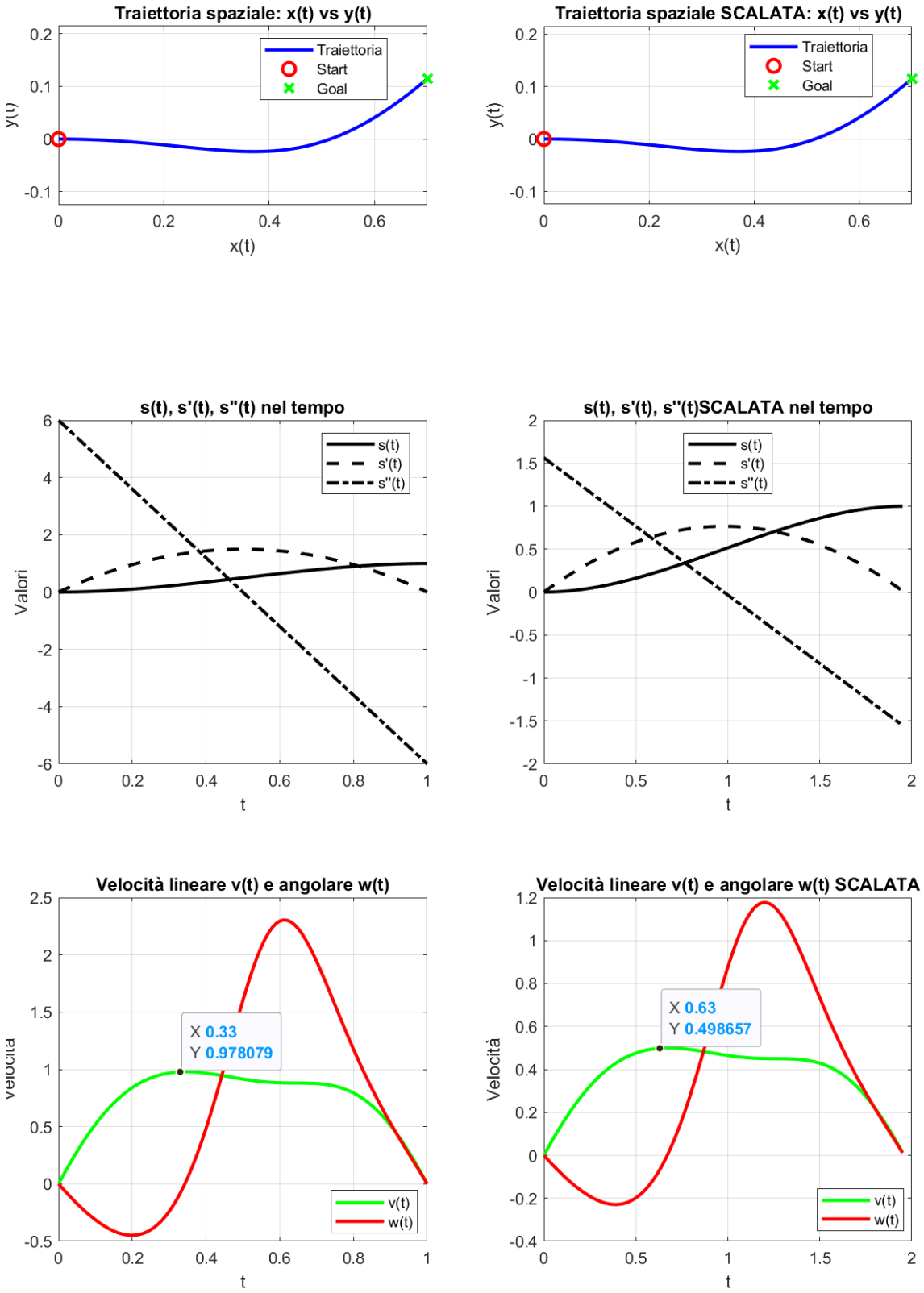


Fig. 2. path planning algorithm for unicycle, scaled and unscaled with final pose qf=[0.7004, 0.1144, 0.7045]

CAPITOLO 2. INPUT/OUTPUT LINEARIZATION CONTROL

2. Given the trajectory in the previous point, implement via software an input/output linearization control approach to control the unicycle's position. Adjust the trajectory accordingly to fit the desired coordinates of the reference point *B* along the sagittal axis, whose distance to the wheel's center it is up to you. Critically comment throughout the report how the results are affected by changing the distance of *B* from the unicycle's reference point. [*Hint: Consider 3-4 cases for the comparison.]*

To implement the input/output linearization control approach for the unicycle's position, the first step is to compute the position of point B (y1,y2), located at a distance b from the wheel center (x,y).

$$y_1 = x + b\cos\theta$$
$$y_2 = y + b\sin\theta$$

Next, we calculate the first derivatives of y_1 and y_1 with respect to time. The values we need to obtain to implent the control are the desired y_{1d} and y_{2d}, so we use $v_d, \omega_d \; and \; \theta_d$

$$\begin{bmatrix}\dot{y}_1\\ \dot{y}_2\end{bmatrix} = \underbrace{\begin{bmatrix}\cos\theta & -b\sin\theta\\ \sin\theta & b\cos\theta\end{bmatrix}}_{T(\theta)}\begin{bmatrix}v\\ \omega\end{bmatrix}$$

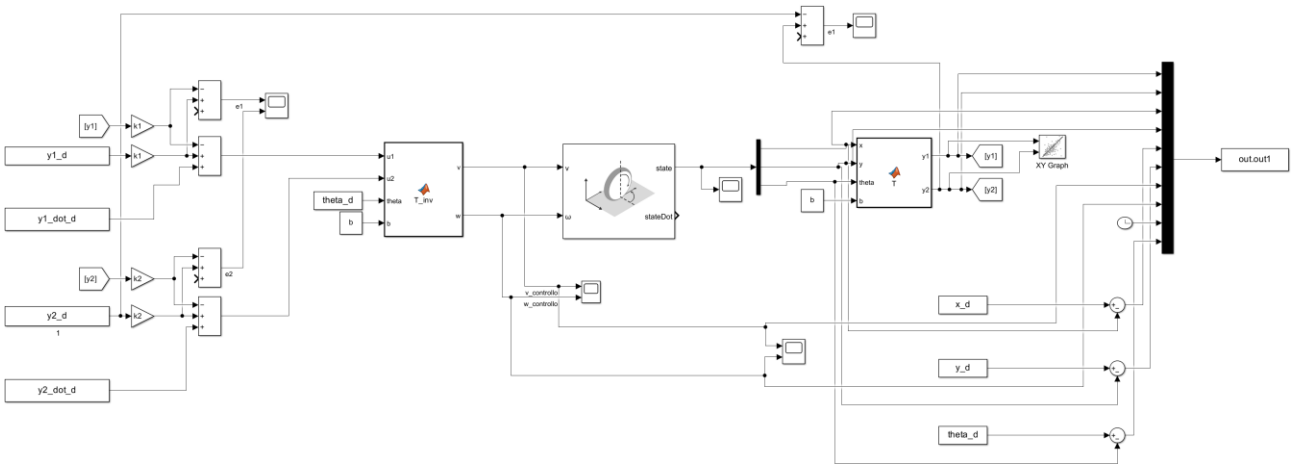
The Simulink model is controlled through the input variables v and ω, which are obtained by inverting matrix T and using two virtual inputs defined as follows:

k1=5;
$$u_1 = \dot{y}_{1,d} + k_1(y_{1,d} - y_1)$$

k2=5;
$$u_2 = \dot{y}_{2,d} + k_2(y_{2,d} - y_2)$$

Unfortunately, this control strategy only regulates the position of point B, while leaving the orientation of the unicycle uncontrolled.

Let us consider the previous case with q_f=[0.7004, 0.1144, 0.7045] and simulation time 1.958sec, for all simulations in this document the radius of the unicycle wheel is chosen equal to 10 cm.



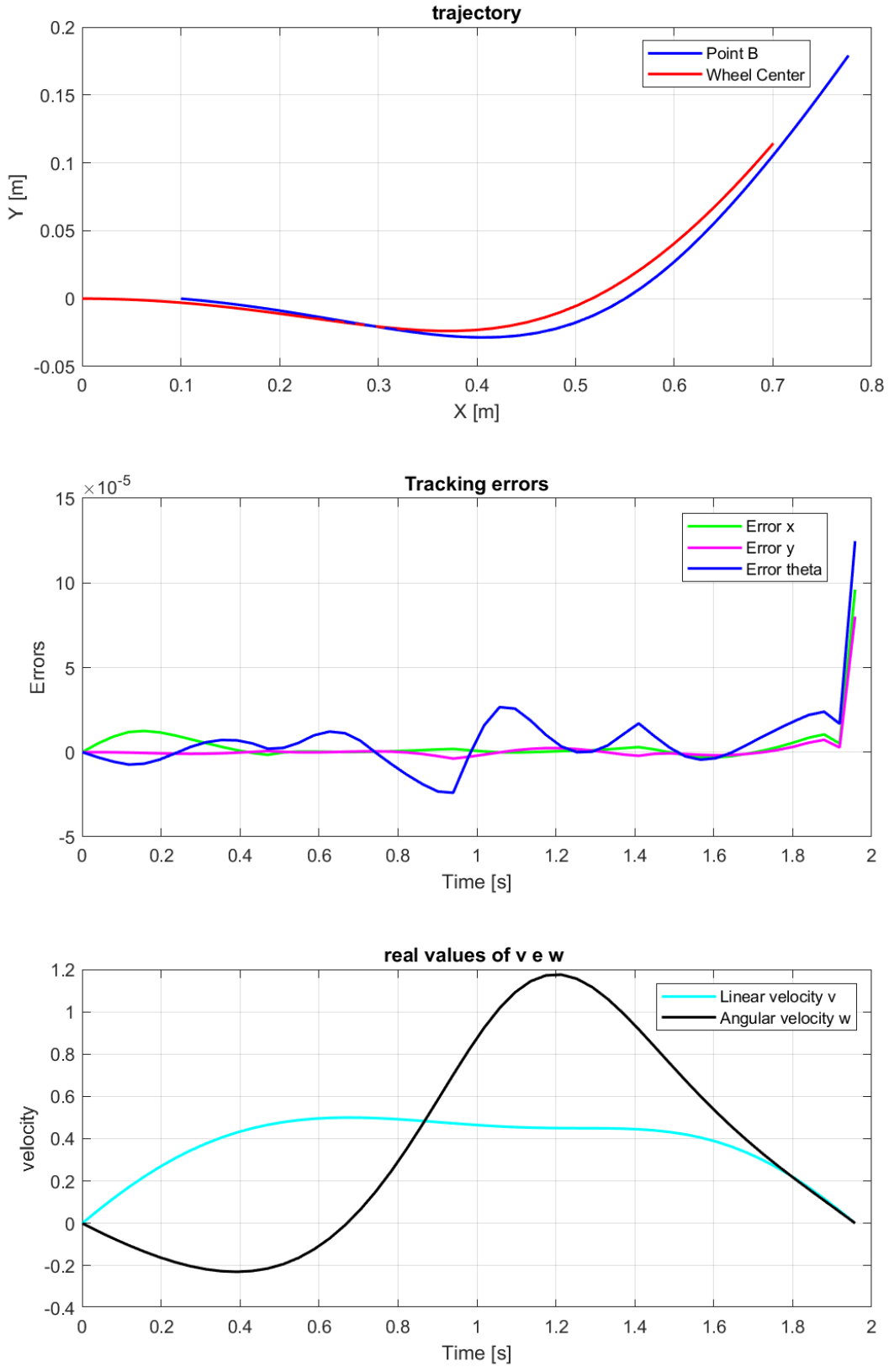


Fig. 3. graphs of the significant variables with the distance of B from the center of the wheel of 10cm

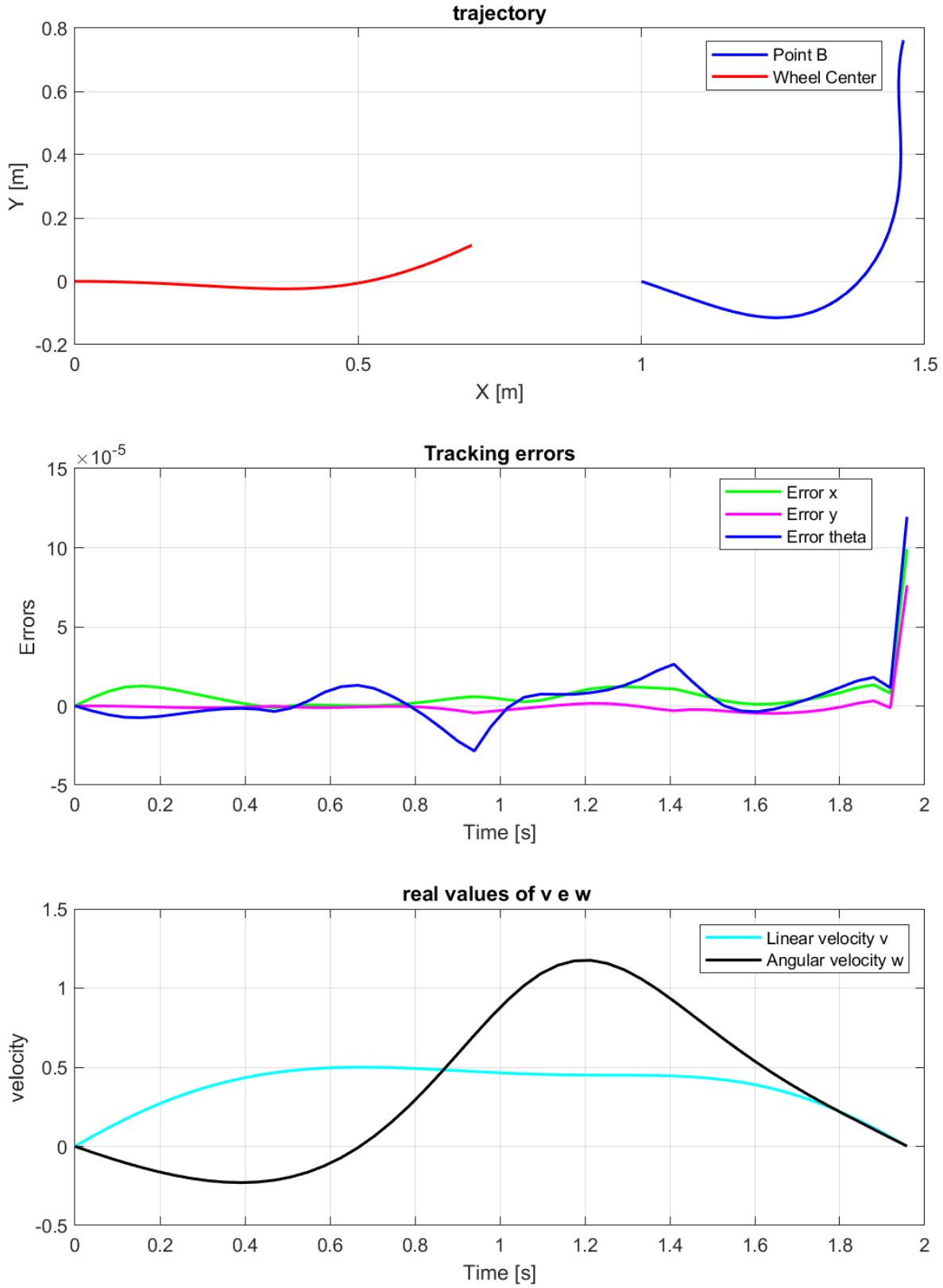


Fig. 4. graphs of the significant variables with the distance of B from the center of the wheel of 100cm

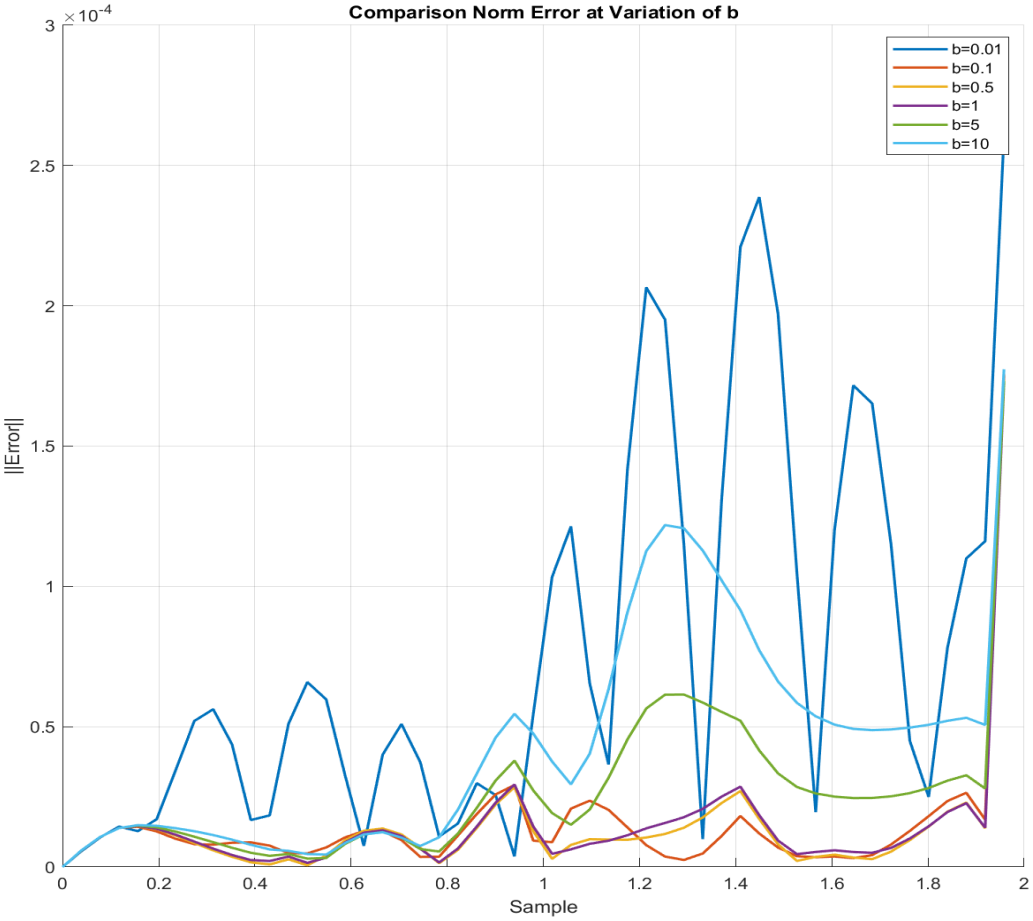


Fig. 5. Comparison Norm Error at variation of b

In the control of a unicycle, the tracking error is evaluated by considering a point B located at a distance b from the wheel's center.

Simulation results show that as the distance b increases, the position error of the wheel relative to the desired trajectory also increases.

In the plot above, the norm of the tracking error in x, y, and θ is shown for different values of the parameter b. It can be observed that the error consistently grows as b increases, although the overall error remains relatively small.

However, an anomalous behavior is observed when b = 0.01. This behavior can be explained by looking at the control law: the control velocities are computed using the inverse of matrix T. The second column of T is directly dependent on b. When b approaches zero , as in the case of b = 0.01 , the determinant of T becomes very small, which leads to numerical instability and inaccuracies during the matrix inversion process.

It is important to note that if b were exactly zero, the matrix T would lose rank and become non-invertible, since its rank would drop to 1.

In conclusion, reducing b generally leads to a lower tracking error. However, if b becomes too small, numerical problems arise due to the near-singularity of T, causing the error to increase again. This explains the abnormal behavior observed for small values of b, such as 0.01.

CAPITOLO 3. BERNOULLI'S LEMINSCATE TRAJECTORY

3. Consider the Bernoulli's leminscate trajectory

$$x_d(t) = \frac{r\cos((\alpha+1)t)}{1+\sin^2((\alpha+1)t)},$$

$$y_d(t) = \frac{r\cos((\alpha+1)t)\sin((\alpha+1)t)}{1+\sin^2((\alpha+1)t)},$$

where α the last digit of your matriculation number, $t \in [0, \frac{4\pi}{\alpha+1}]$, and consider two different cases with two arbitrary values for $r > 0$ that should differ of at least one order of magnitude. For both cases, design via software the linear and (almost) nonlinear controllers for a unicycle. Suppose that the unicycle at the time $t = 0$ starts from a random position generated by the code within 0.5 m from the desired initial one. Through relevant plots, critically compare the results in the report.

Considering the Bernoulli's leminscate trajectory, $x_d(0) = r$, $y_d(0) = 0$
The initial position of the system will therefore be

$$x(0) = r + rand_2 * \cos(rand_1)\,,\; y(0) = rand_2 * \sin(rand_1)$$

where $rand_1 \in [0,2\pi]$ and $rand_2 \in [0,0.5]$
The last digit of my matriculation number is 1, so α =1 and the simulation will be simulated for 2π seconds.

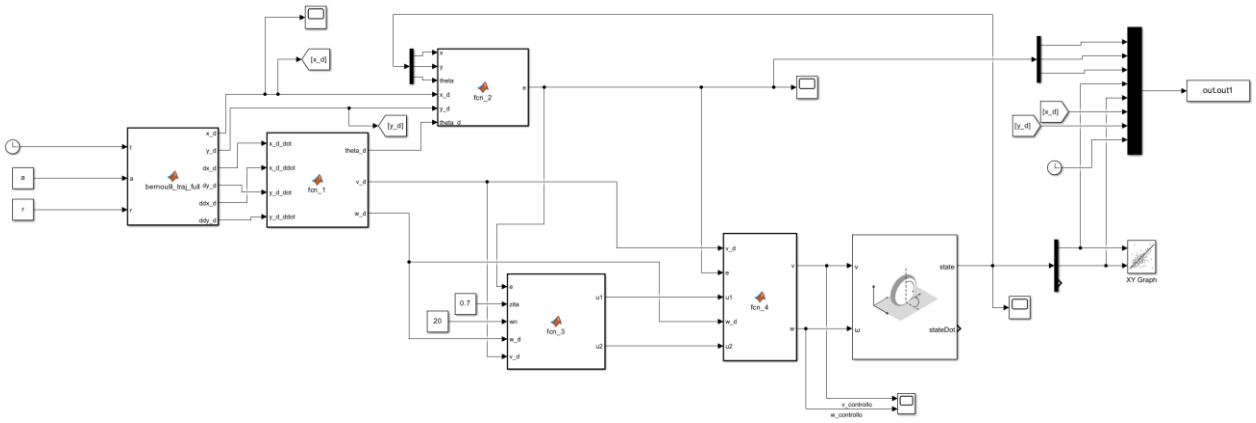


Fig. 6. simulink scheme

The system is divided into five main blocks:

1. **bernoulli_tarj_full**
   This block computes, at each time step, the desired position and its derivatives (velocity and acceleration).
   The derivatives were calculated mathematically, since using the first and second order derivative block, the noise introduced made the results considerably worse.
2. **fcn_1**
   This function calculates the desired values of linear velocity v_d, angular velocity ω_d, and the desired orientation angle θ_d.

3. **fcn_2**
   This block computes the error between the current and desired states, using a rotation matrix to express the error in the robot's reference frame. The errors shown in the graphs are related to this new reference system.
4. **fcn_3**
   Using the error values and ω_d and v_d this function computes the two virtual control inputs. This is the only block that is modified when using Control-based on approximate linearization or almost non-linear control.

$$u_1(t) = -k_1 e_1(t)$$
$$u_2(t) = -k_2(t) e_2(t) - k_3 e_3(t)$$

$$k_1 = 2\zeta\omega_n$$
$$k_2 = \frac{\omega_n^2 - \omega_d(t)}{v_d(t)}$$
$$k_3 = 2\zeta\omega_n$$

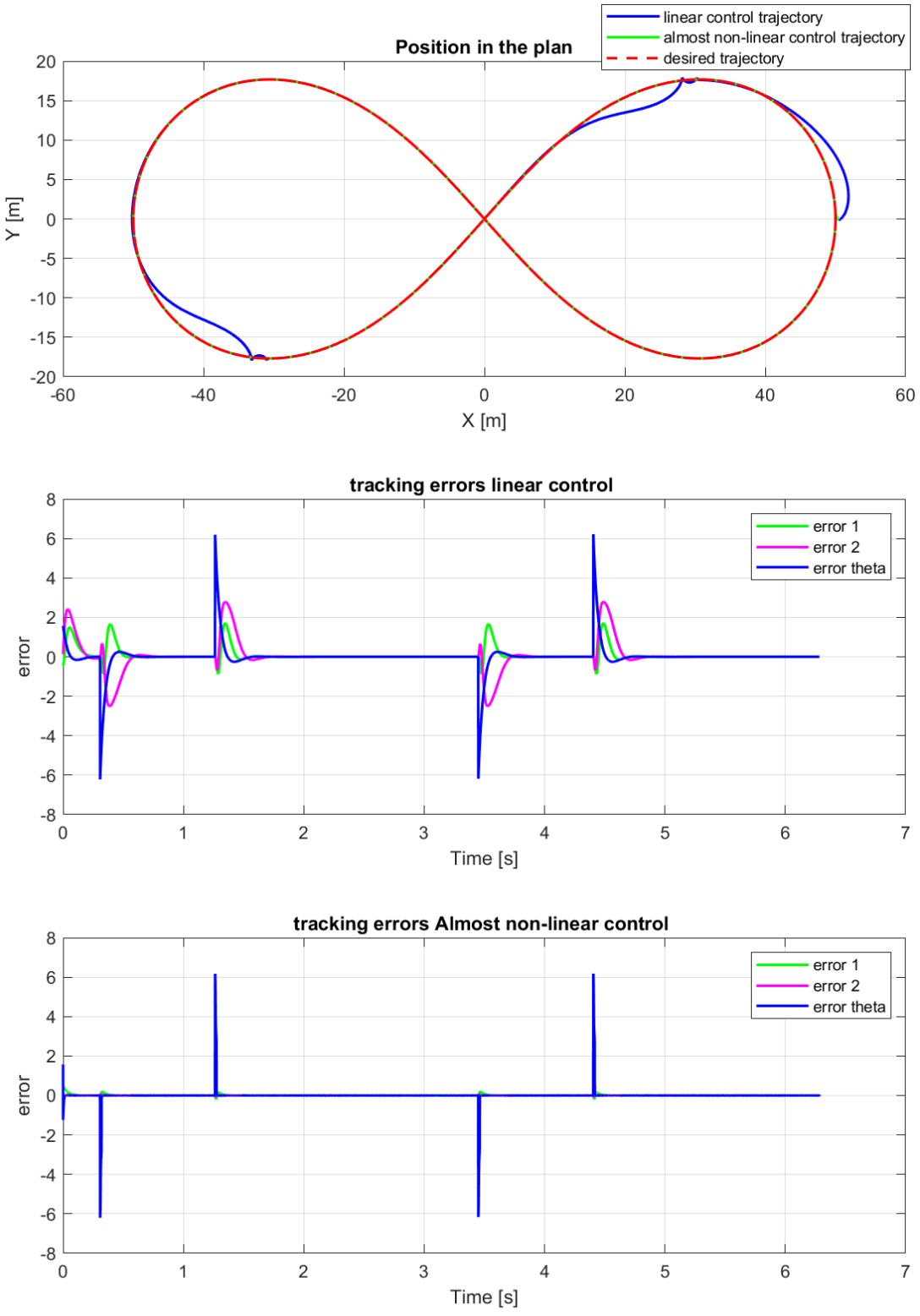
(2)
$$u_1(t) = -k_1(v_d(t), \omega_d(t)) e_1(t)$$
$$u_2(t) = -k_2 v_d(t) \frac{\sin e_3(t)}{e_3(t)} e_2(t) - k_3(v_d(t), \omega_d(t)) e_3(t)$$

$$k_1 = 2\zeta\omega_n$$
$$k_2 = \omega_n^2$$
$$k_3 = 2\zeta\omega_n$$

(ζ = 0.7 and ω_n = 20, selected through trial and error)

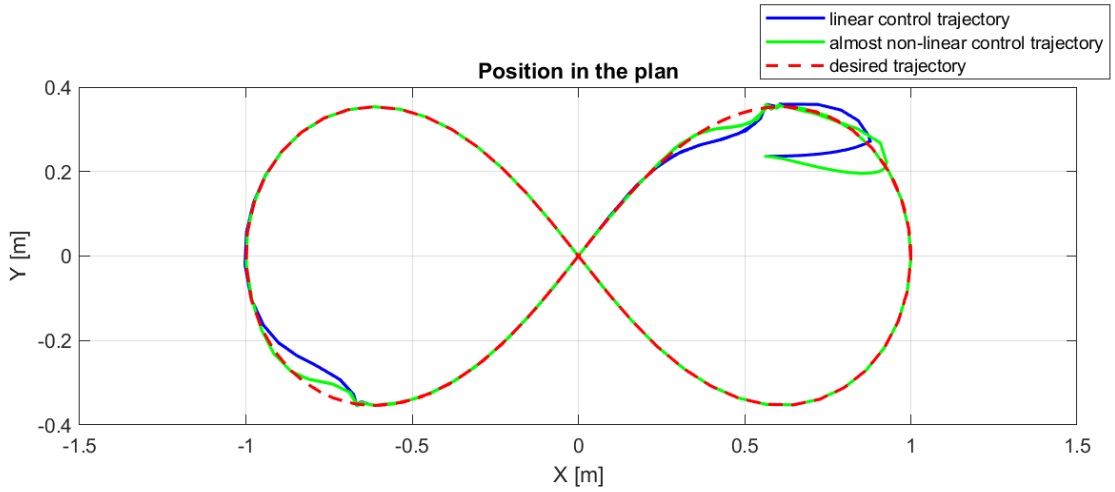
   In the Control-based on approximate linearization , the controller parameters were selected by analyzing the eigenvalues of the linearized model, ensuring system stability and good dynamic response.
   In the Almost nonlinear case, the gains k_1, k_2, k_3 were chosen using a similar approach, but with an important difference: the dependency of k_2 on the desired linear and angular velocities was removed; while even though the values of k_1 k_3 could depend on ω_d and v_d it was preferred to leave them constant.
   As a result, the parameters ζ (damping ratio) and ω_n (natural frequency), which had a clear mathematical meaning in the linear case, lose their theoretical significance here.
   Nevertheless, simulation results show that this design choice still leads to improved system performance.
5. **Fcn_4**
   Finally, this block gathers all the previously computed information and calculates the actual control commands to be sent to the plant.

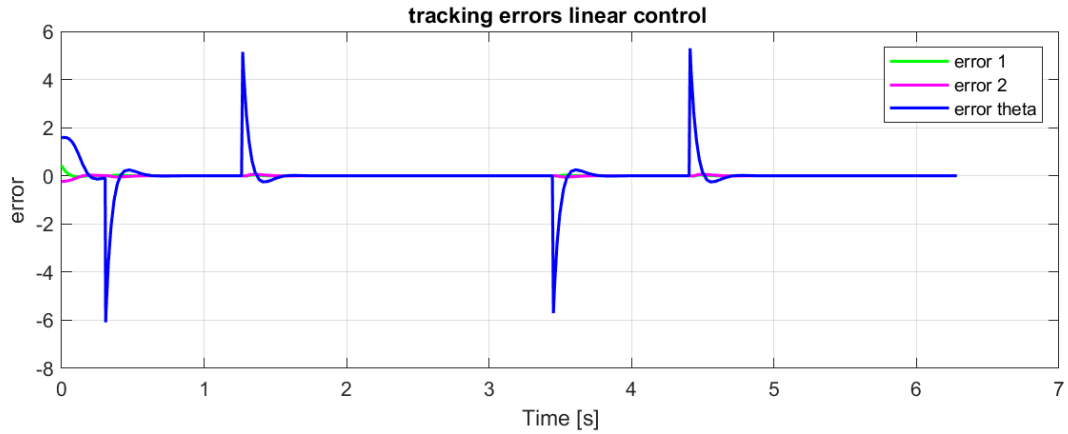
The error shown in the graphs is considered to be rotated in the current direction.
The simulations performed with a 50-meter radius were carried out solely for comparison purposes. During these tests, the unicycle reached speeds of up to 150 m/s , values that are highly unrealistic, especially considering the wheel's radius is only 10 cm and the reference trajectory contains a large number of curves.

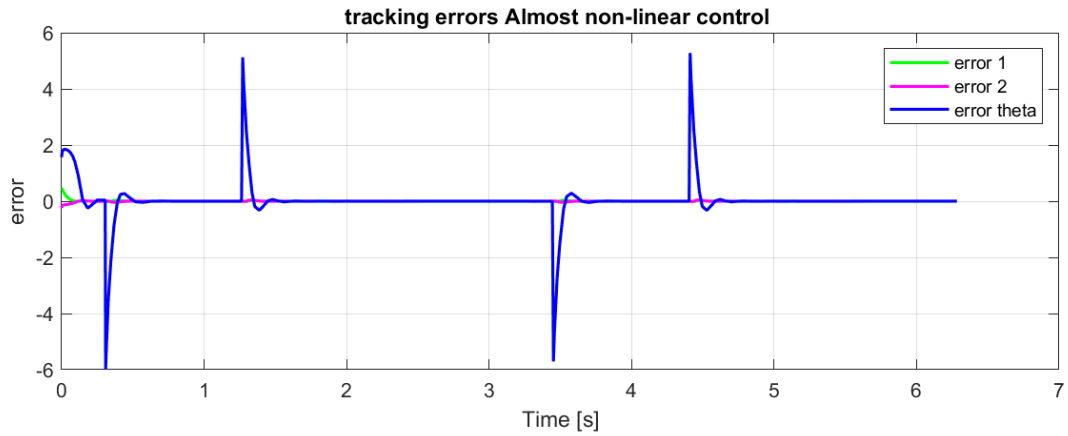



```
Maximum distance between Trajectory linear control and Reference: 3.2758 m
Maximum distance between Trajectory Almost non-linear and Reference: 0.2267 m
```

Fig. 7. Performance comparison between Almost-non-linear and linearisation-based control with R=50








```
Maximum distance between Trajectory linear control and Reference: 0.1541 m
Maximum distance between Trajectory Almost non-linear and Reference: 0.0774 m
```

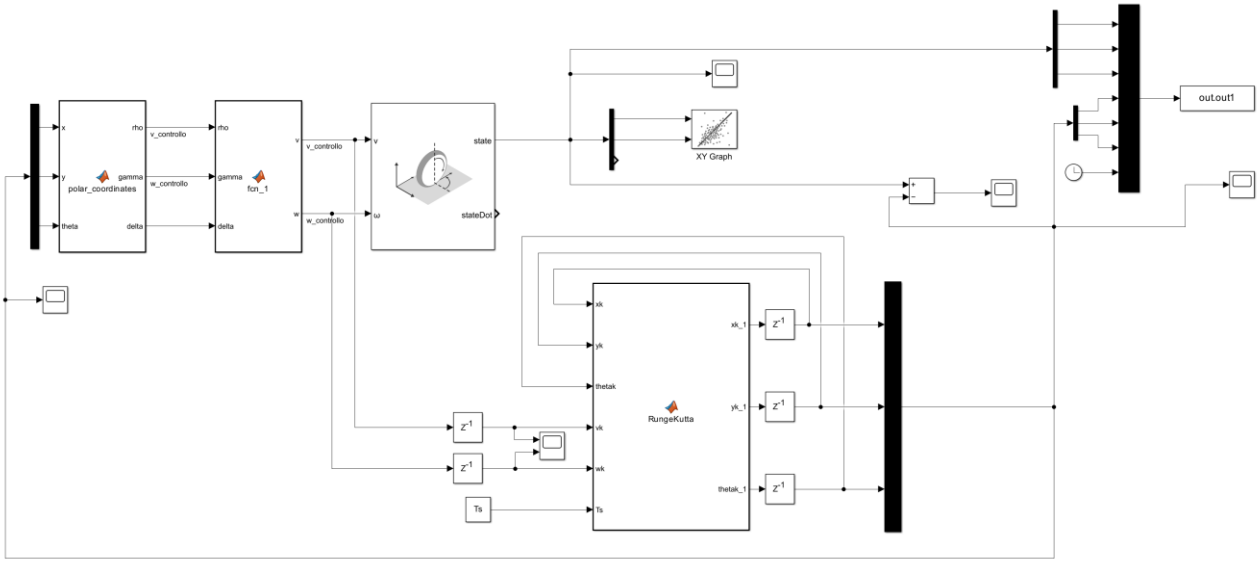
Fig. 8. Performance comparison between Almost-non-linear and linearisation-based control with R=1

It is worth noting that in order to calculate the maximum error along the trajectory, the initial values were eliminated, in fact, at the beginning the error could be 0.50m.
Since the time allocated to complete the trajectory remains unchanged regardless of the chosen radius, it is expected that for a trajectory with a 50-meter radius, the unicycle reaches significantly higher speeds compared to the case with a 1-meter radius; This leads to an increase in error when the control gains remain fixed. For the sake of clarity and practicality, the plots of linear and angular velocity have not been included, although these values were monitored during the simulations.
During the 50-meter radius trajectory, the position error (which can be computed either using the coordinates $x_d - x \; and \; y_d - y$ or using $e_1 \; and \; e_2$, as the change of reference frame does not affect the notion of distance) exceeds 3 meters. Despite this, the linearized controller manages to drive the system back to zero position error, highlighting that the attraction region of the overall system, under linear control, is quite large.
It is worth noting that, in theory, linear control validation is guaranteed only within a neighborhood of the equilibrium point in the nonlinear system, but the actual size of this region is generally unknown.
In both scenarios, the Almost Non-Linear control strategy demonstrated better performance compared to the Control-based on approximate linearization, achieving a tracking error up to 15 times smaller than the one observed with the linear approach.
It is important to note, however, that both control strategies share a common limitation: they cannot directly control the desired orientation θ, and require persistent trajectories (a condition that is satisfied within the time interval $[0,2\pi]$).
The lack of direct control over the orientation leads to noticeable error spikes (of amplitude 2π) during trajectory tracking. This behavior is mainly due to the fact that θ depends on the arctan2(y/x), which is discontinuous. As x and y naturally change sign along the path, the angle experiences abrupt transitions, making it impossible to maintain a constant zero error on the orientation.

CAPITOLO 4. UNICYCLE POSTURE REGULATOR BASED ON POLAR COORDINATES

4. Implement via software the unicycle posture regulator based on polar coordinates, with the state feedback computed through the Runge-Kutta odometric localization method. Starting and final configurations are $q_i = [x_i \quad y_i \quad \theta_i]^T = [\alpha + 1 \quad 1 \quad \pi/4]^T$, with α the last digit of your matriculation number, and $q_f = [x_f \quad y_f \quad \theta_f]^T = [0 \quad 0 \quad 0]^T$.

The last digit of my matriculation number is 1,so the initaial position $q(0) = [2\ 1\ \pi/2]$



To implement this control algorithm, four MATLAB Function blocks were used:

- **Polar Coordinates**: takes as input the estimated current state of the robot and the desired pose (in this case, the origin), and computes the parameters ρ , γ, and δ.
- **Fcn_1**: uses these parameters to compute the control inputs, i.e., the desired linear velocity v and angular velocity w. The parameters used for the control were chosen via the traial e error technique as k1=3, k2=3 and k3=1;
- **Runge-Kutta**:
  The state estimation was performed using an odometric localization method based on the second order Runge-Kutta appoximation.
  The odometry computation was implemented with a sampling time of Ts. To compute the next state, the following information was required:
  - the previously estimated state, appropriately delayed;
  - the initial state conditions, which were set within the delay block;
  - the linear and angular velocities of the unicycle, also retrieved with a delay.

  Although the Runge-Kutta method provides higher accuracy compared to simpler approaches such as forward Euler integration, the solution is still affected by a certain amount of drift.
  In fact, while the estimated state converges exactly to zero, the actual state (as retrieved

from the unicycle model block) shows a steady-state error, which varies depending on the chosen sampling time,leaving gains K unchanged:

- for Ts = 0.02 s → error: [-0.06, 0.06, 0.04]
- for Ts = 0.01 s → error: [-0.03, 0.02, 0.01]
- for Ts = 0.001 s → error: [-0.003, 0.002, 0.002]
- Finally, when using Ts = 0.1 s, the system turns out to be unstable, highlighting the importance of the sampling time in ensuring both estimation accuracy and system stability.

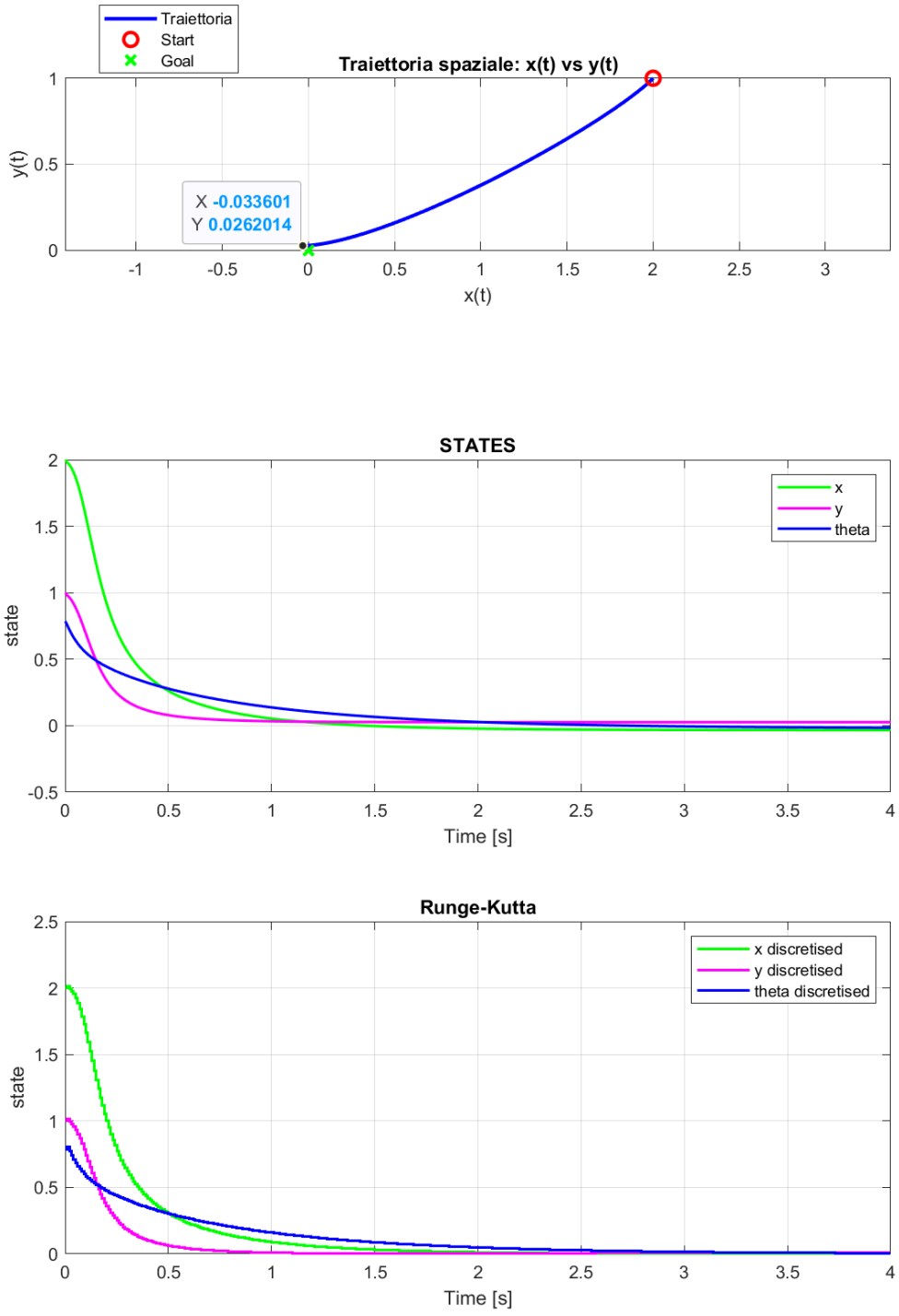


Fig. 9. unicycle posture regulator based on polar coordinates Ts=0.01