RC2 Project

Tommaso Tubaldo
Riccardo Zattra
Davide Molinaroli

Università degli Studi di Padova
Dipartimento di Ingegneria dell'Informazione

April 4, 2025

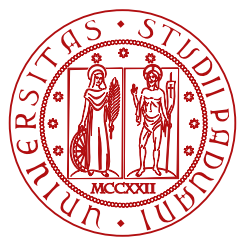


Table of contents

Table of contents

Problem Description and Motivations

For the unicycle presented in class:

- design one or more controllers to follow a chosen desired trajectory
- avoid obstacles during the path
- at the end of the maneuver park in a chosen position with a specific orientation

Motivation:

- search and rescue vehicles
- automated warehouses
- farming automation (autonomous tractors)
- ...

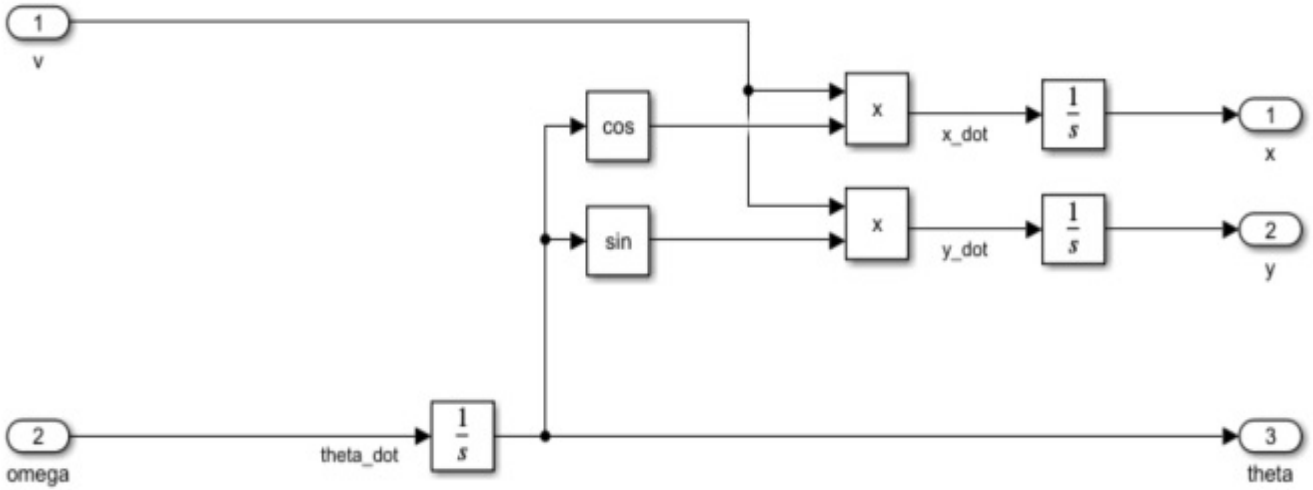
Table of contents

Trajectory tracking via state error feedback - Unicycle model

As presented in class the model is:

$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta)$$
$$\dot{\theta} = \omega$$

- x, y: planar coordinates
- θ: orientation of the robot w.r.t the x - axis,
- v: linear velocity (m/s),
- ω: angular velocity (rad/s).

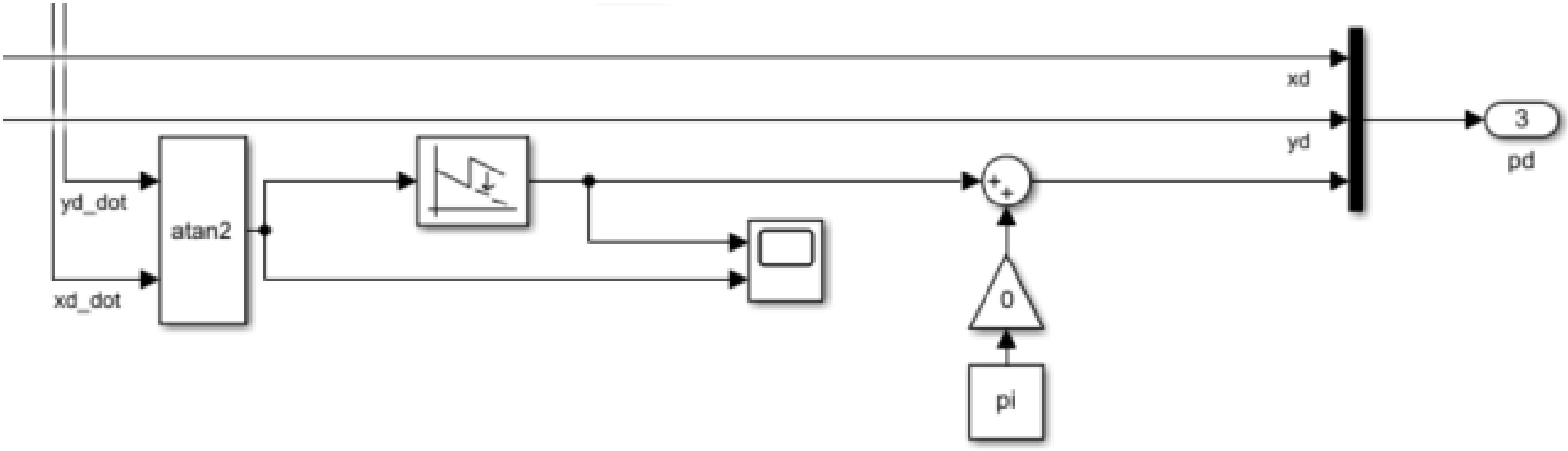


Trajectory tracking via state error feedback - Differential flatness

If we define the output as the desired trajectory in cartesian coordinate then:

$$\theta_d(t) = atan2(\dot{y}_d(t), \dot{x}_d(t)) + k\pi \quad k \in \{0, 1\}$$

- $k = 0 \rightarrow$ forward motion
- $k = 1 \rightarrow$ backward motion



Trajectory tracking via state error feedback - Controller

Idea behind the controller:

- Evaluate the error in body frame
- Write the dynamics of it
- After a change of inputs, linearize the dynamics around the desired trajectory
- Select as input $u = Ke$ where $u \in \mathbb{R}^2, K \in \mathbb{R}^{2\times 3}$, $e \in \mathbb{R}^3$

$$K = \begin{bmatrix} -k_1 & 0 & 0 \\ 0 & -k_2 & -k_3 \end{bmatrix}$$

Which leads to the following error dynamic:

$$\dot{e}(t) = \begin{bmatrix} -k_1 & \omega_d(t) & 0 \\ \omega_d(t) & 0 & v_d(t) \\ 0 & -k_2 & -k_3 \end{bmatrix} e(t)$$

Trajectory tracking via state error feedback - Trajectory

Let $a \in \mathbb{R}^+ \setminus \{0\}$ and $\xi \in (0, 1)$, selecting:

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

and selecting a circular trajectory with constant $\omega_d(t)$ then the error dynamic becomes time invariant.

Let $R \in \mathbb{R}^+ \setminus \{0\}$ the radius of the circle and ω_d the desired angular velocity, the parametrization is given by:

$$\begin{cases} x_d(t) = R\cos(\omega_d t) \\ y_d(t) = R\sin(\omega_d t) \end{cases}$$

```
% First quarter of circle
if(flag == 1)
    enable_traj = 1;
    enable_park = 0;
    x = r1*cos(omega*t);
    y = r1*sin(omega*t);
end
```

Trajectory tracking via state error feedback - Control law implementation

With the above choice of the coefficients of the controller, namely:

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

In k_2 we are dividing by $v_d(t) \rightarrow$ particular care must be taken in the implementation of the control law to avoid divisione by zero.

Simulink implementation:

```
function inputs = fcn(vd,wd,eb)
d = 0.4;%csi
a = 8;
k1 = 2*d*a;
if abs(vd) < 1e-4
    if vd >= 0
        vd = 1e-4;
    else
        vd =-1e-4;
    end
end
k2 = (a^2- wd^2)/vd;
k3 = k1;
k = [-k1,0,0;0,-k2,-k3];
inputs = k*eb;
```

Trajectory tracking via state error feedback - eigenvalues

By selecting k_1, k_2, k_3 as in the previous slide we obtain the following eigenvalues of the error dynamics:

- $\lambda_1 = -2\xi a$
- $\lambda_2 = -\xi a + ja\sqrt{1-\xi^2}$
- $\lambda_3 = -\xi a - ja\sqrt{1-\xi^2}$

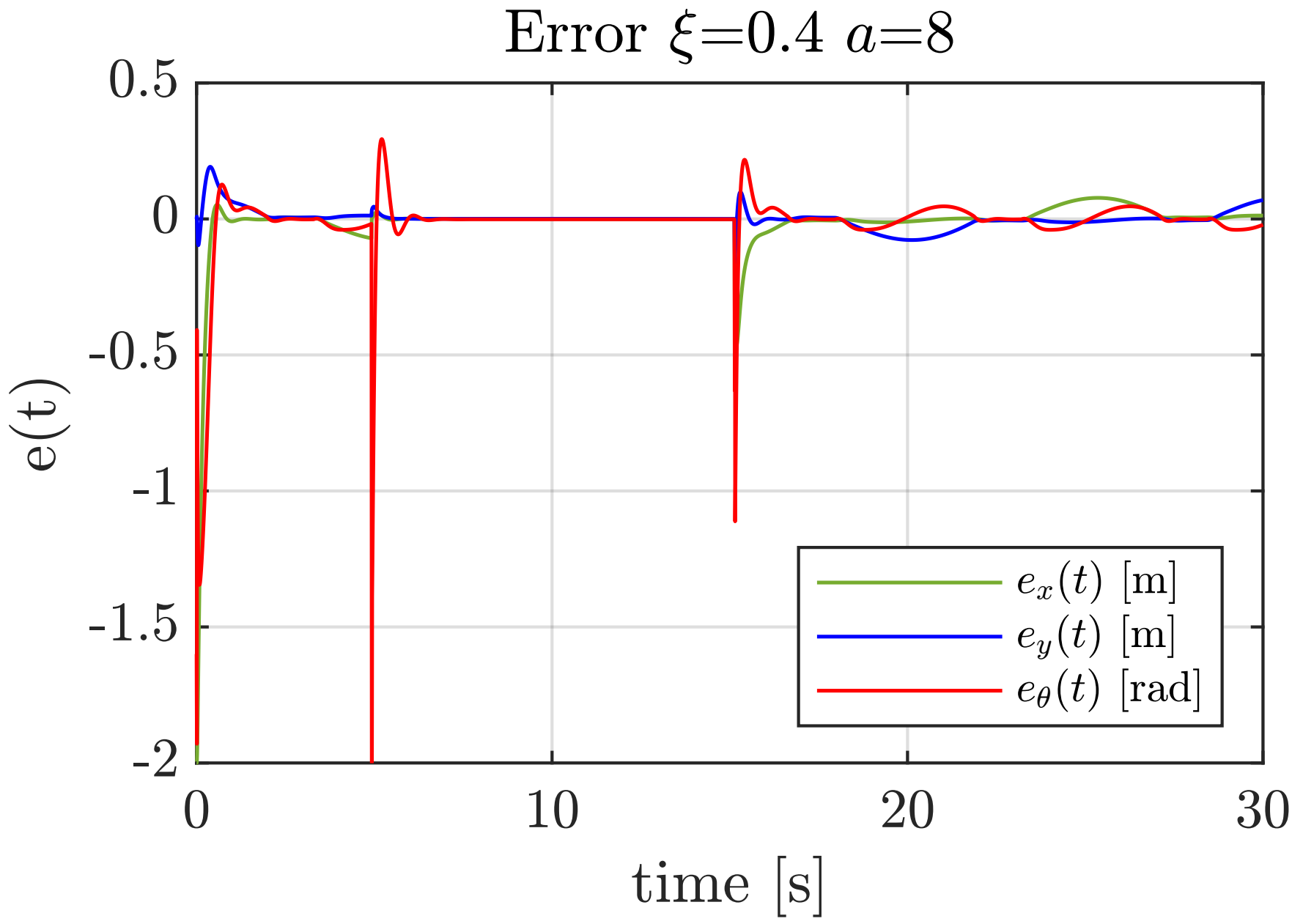
The error at time t will be:

$$e(t) = T \begin{bmatrix} e^{\lambda_1 t} & 0 & 0 \\ 0 & e^{\lambda_2 t} & 0 \\ 0 & 0 & e^{\lambda_3 t} \end{bmatrix} T^{-1} e(0)$$

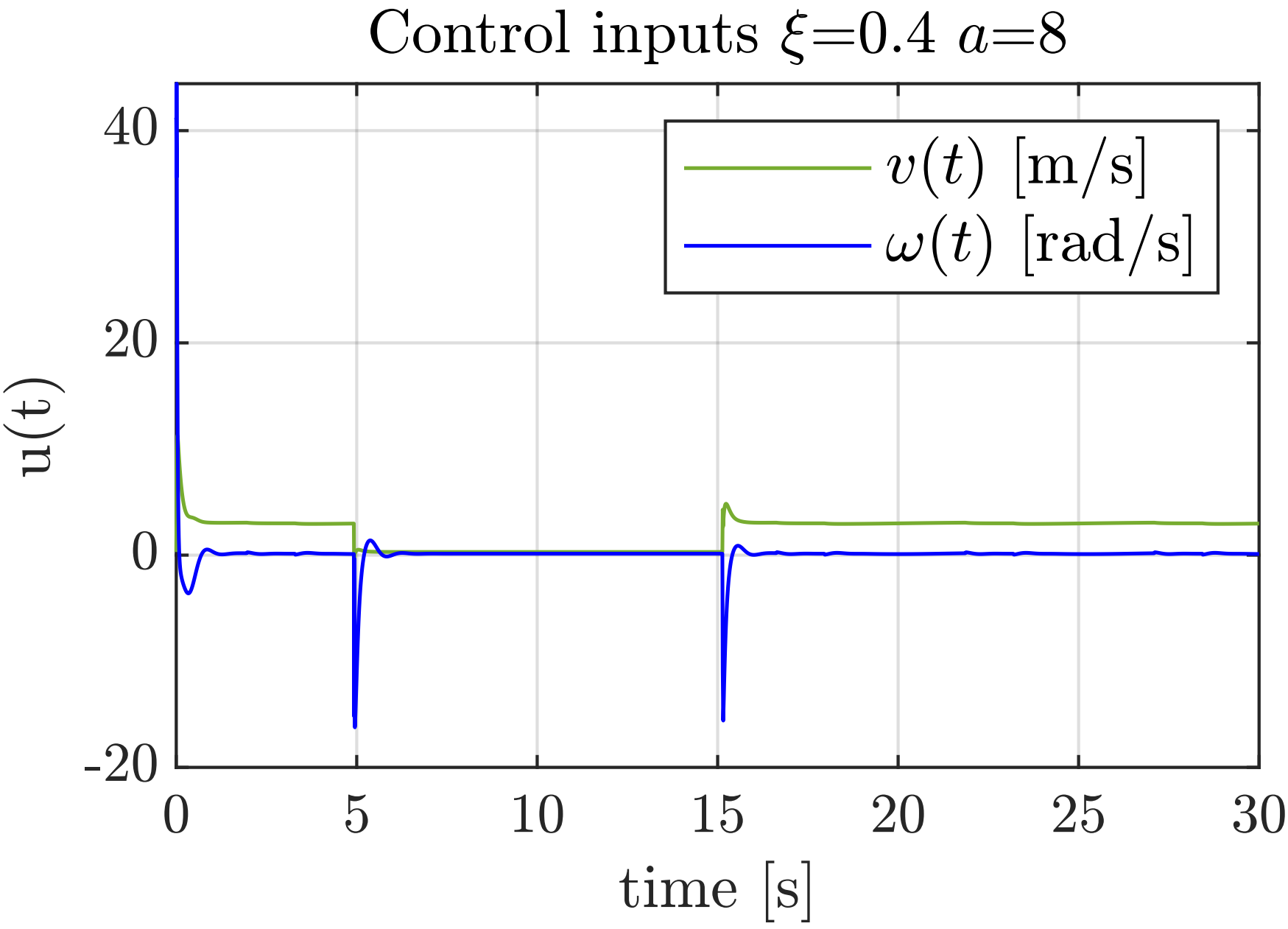
where $T \in \mathbb{R}^{n \times n}$ change of basis.

Note: λ_1 is two times faster than λ_2 and λ_3

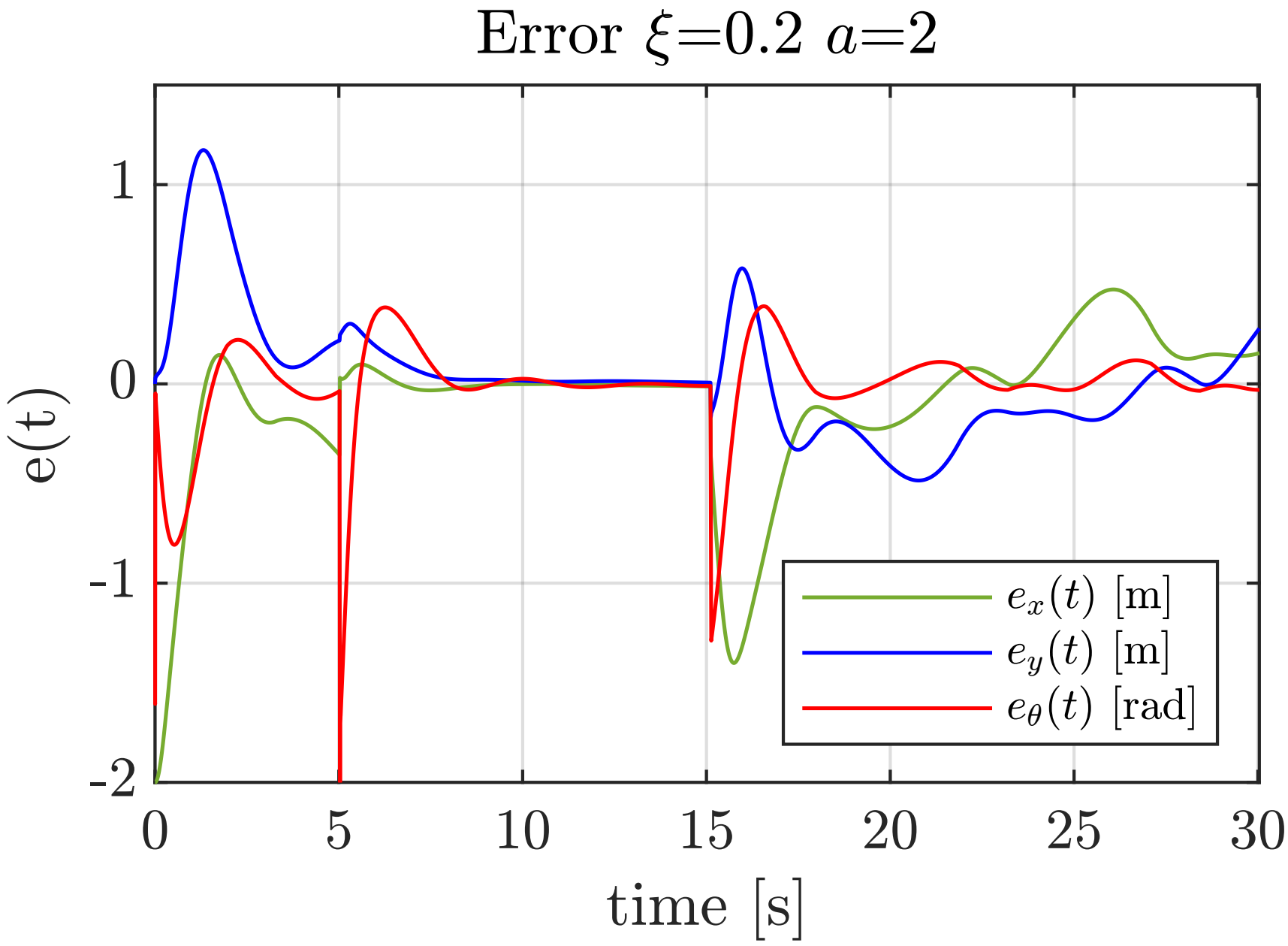
Trajectory tracking via state error feedback - Error



Trajectory tracking via state error feedback - Input



Trajectory tracking via state error feedback - Error



Trajectory tracking via state error feedback - Input

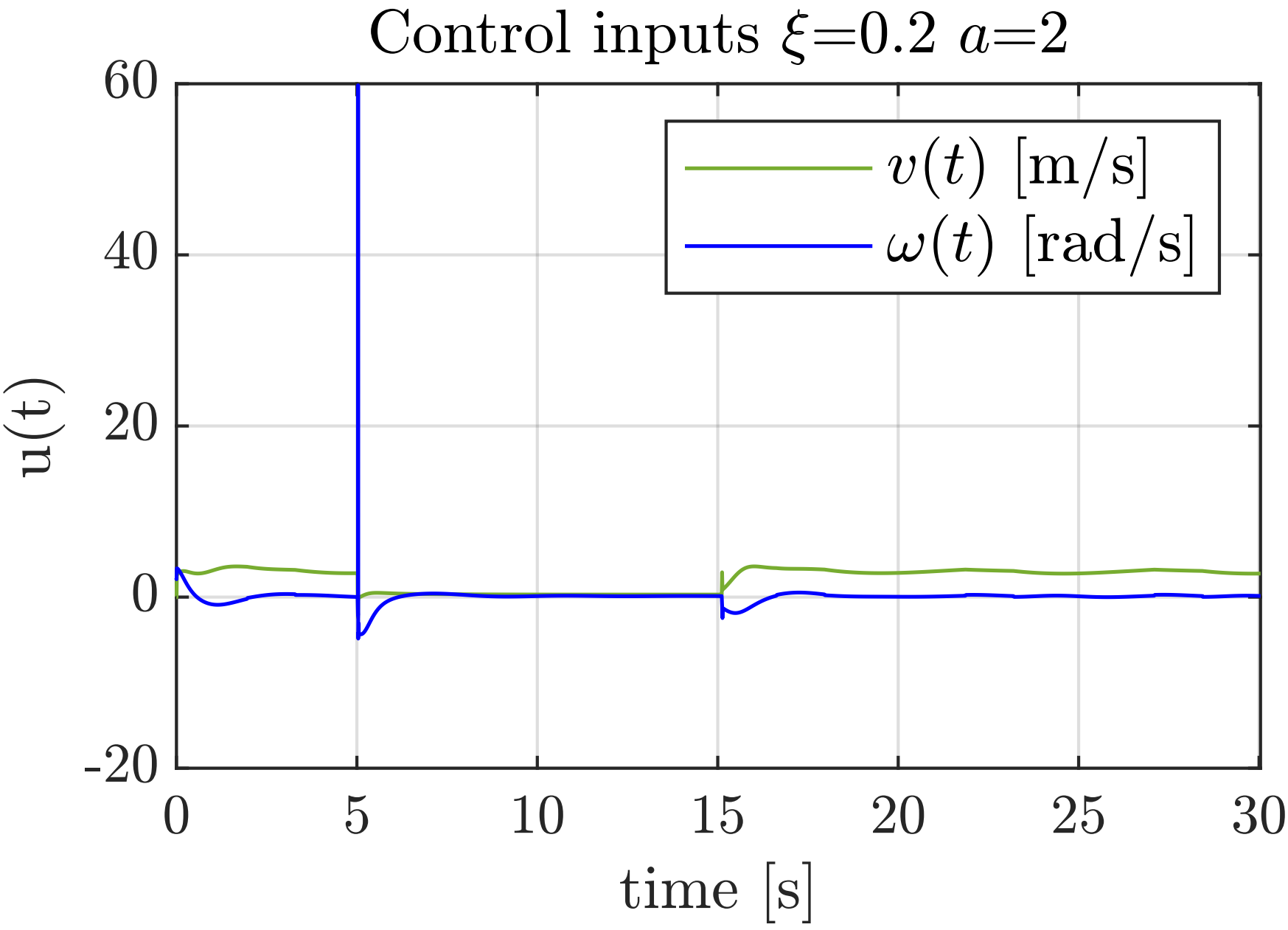


Table of contents

Obstacle avoidance procedure - Assumptions

In order to address this task we made the following assumptions:

- The position of the obstacle is fixed in time ($x_{obs} = 0, y_{obs} = 10$)
- We suppose to know the cartesian coordinates of the unicycle $\rightarrow$ it is possible to measure the distance between the unicycle and the obstacle

If the distance is below a specified threshold the unicycle enter the obstacle avoidance phase, in the following the simulink implementation:

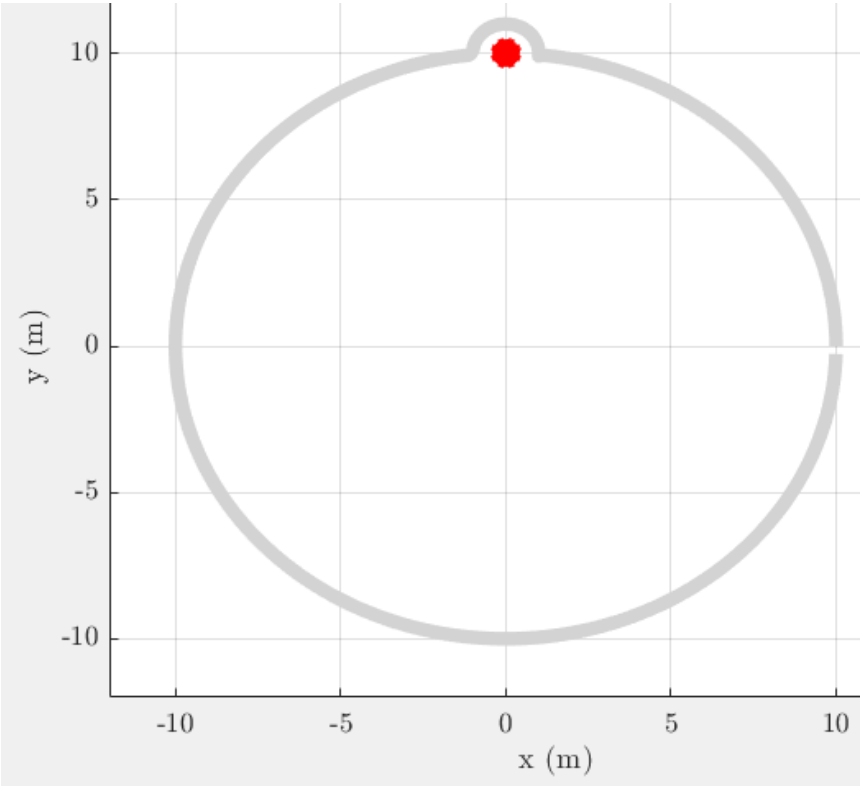
```
if(abs(p(1))<1 && abs(p(2) - r1)<1 && flag==1)
    enable_traj = 1;
    enable_park = 0;
    flag = 2;
end

% Obstacle avoidance
if(flag==2)
    enable_traj = 1;
    enable_park = 0;
    x = r2*cos(omega*(t-pi/2));
    y = r1 + r2*sin(omega*(t-pi/2));
end
if(abs(p(1) + r2)<0.1 && abs(p(2) - r1)<0.1 && flag==2)
    enable_traj = 0;
    enable_park = 1;
    flag = 3;
end
```

Obstacle avoidance procedure - Trajectory

Once the obstacle is detected:

- design a new desired trajectory to avoid the obstacle (a semi circle)
- fed it into the differential flatness block
- once the maneuver is concluded switch back to the original trajectory

Graphically the overall trajectory is the following one:

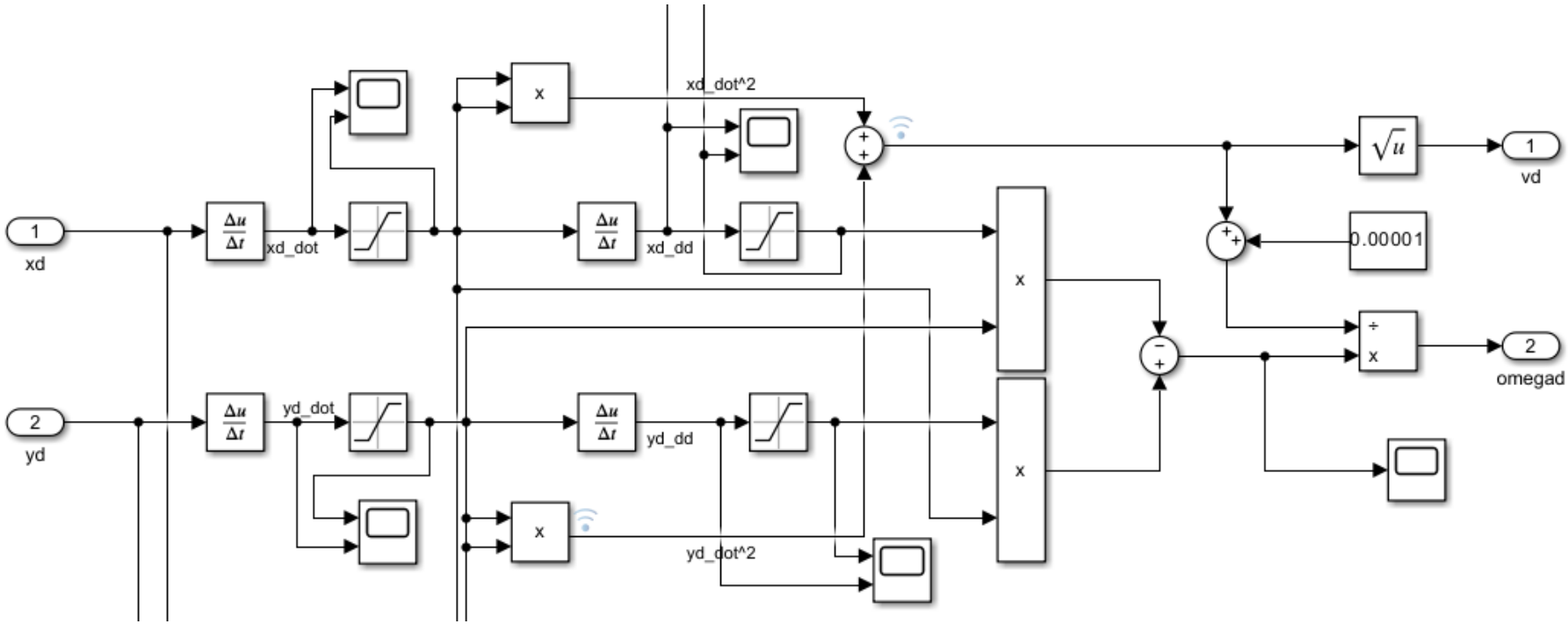


Obstacle avoidance procedure - Switching trajectories

The following problem was encountered:

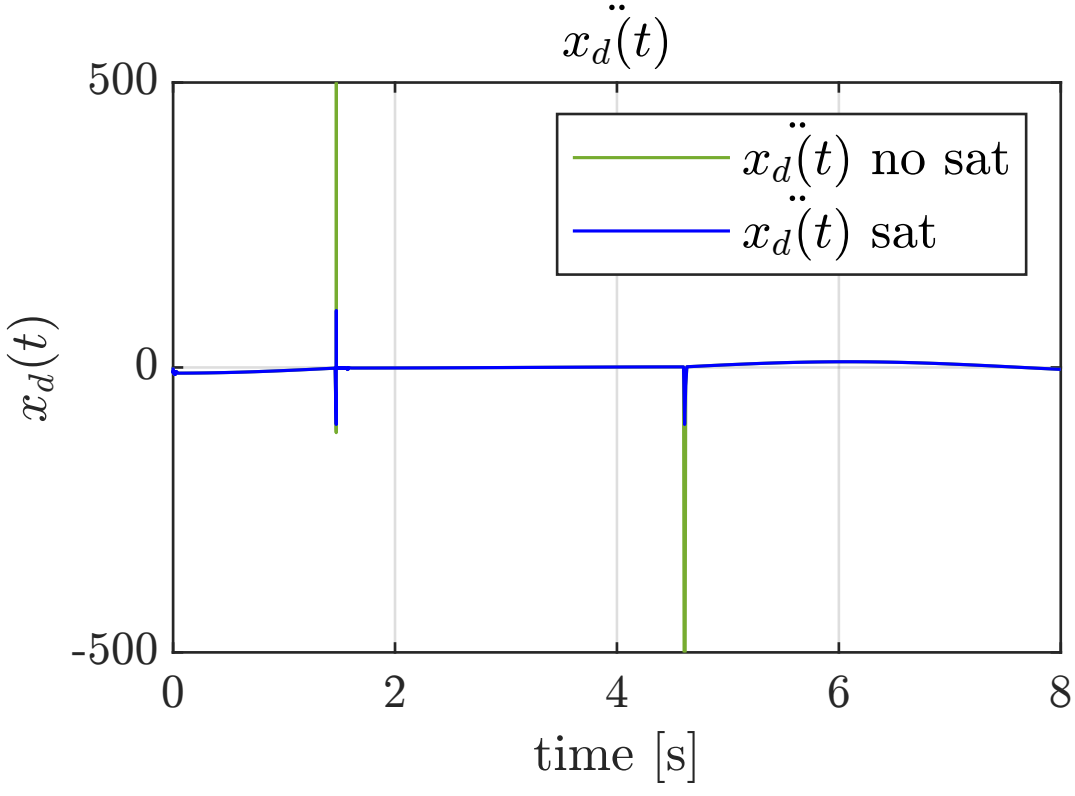
- Switching between different trajectories creates an overall discontinous trajectory fed into the differential flatness block
- This create spikes in $x_d(t), \dot{x}_d(t), y_d(t), \dot{y}_d(t) \rightarrow$ the unicycle will not act properly

Solution $\rightarrow$ add saturation blocks in the differential flatness:



Obstacle avoidance procedure - Switching trajectories solution

In the following a graph reporting the comparision of $\ddot{x}_d(t)$ with and without saturation:



Obstacle avoidance procedure - Results

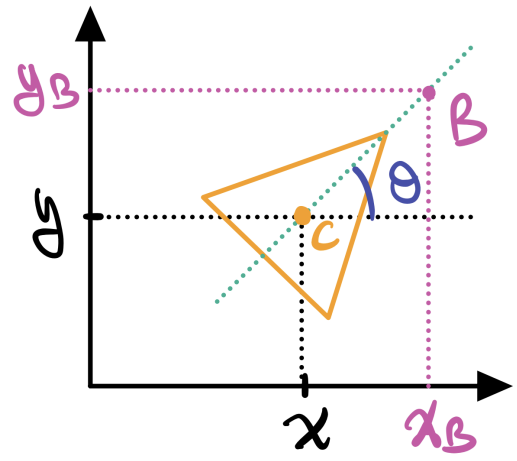
Table of contents

Trajectory Tracking via Output Feedback: B-Point Controller

Idea behind the controller:

- define an invertible map between input and output in order to linearize the system:

$$\begin{cases} x_B = x + b\cos\theta \\ y_B = y + b\sin\theta \end{cases}$$



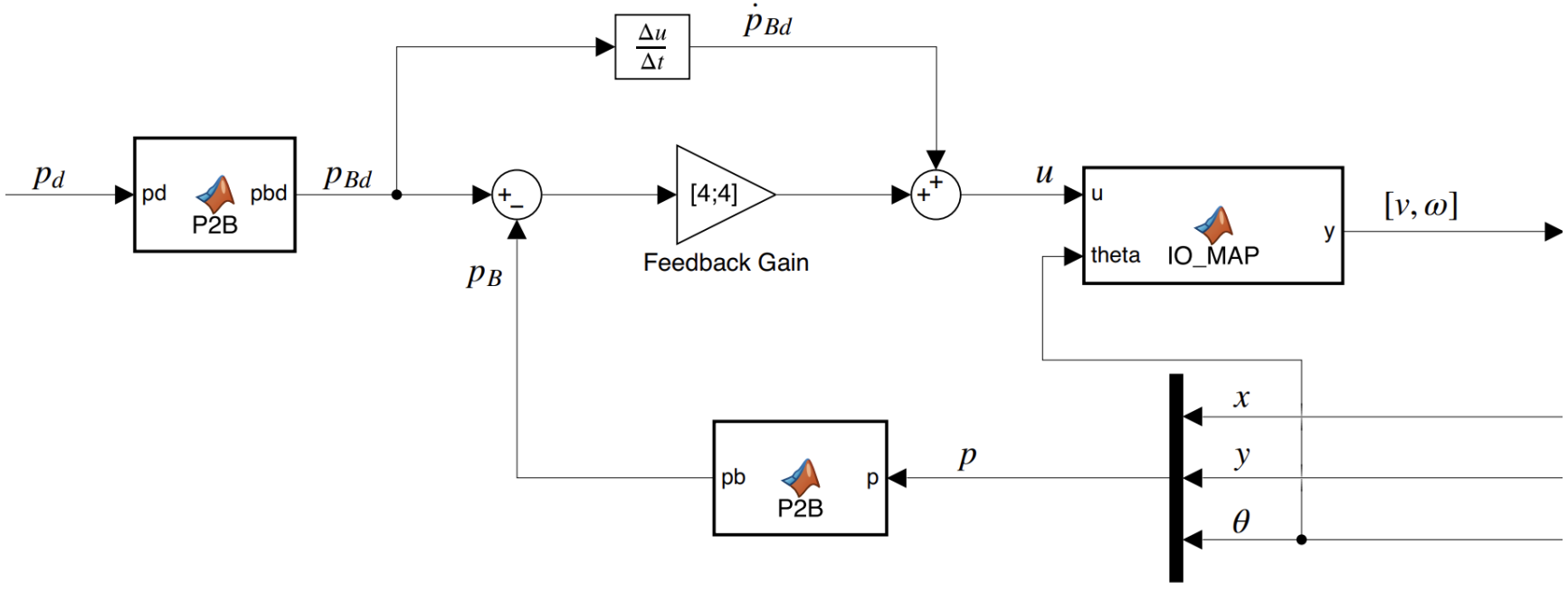
I/O Map: $$\begin{bmatrix} v \\ \omega \end{bmatrix} = \frac{1}{b}\begin{bmatrix} b\cos\theta & b\sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad b \neq 0$$

System Dynamics: $$\begin{cases} \dot{x}_B = u_1 \\ \dot{y}_B = u_2 \\ \dot{\theta} = -\frac{\sin\theta}{b}u_1 + \frac{\cos\theta}{b}u_2 \end{cases}$$

Trajectory Tracking via Output Feedback: B-Point Controller

- choose a suitable value for b, *e.g.* $b = 0.1\,m$
- adopt the following linear control law:

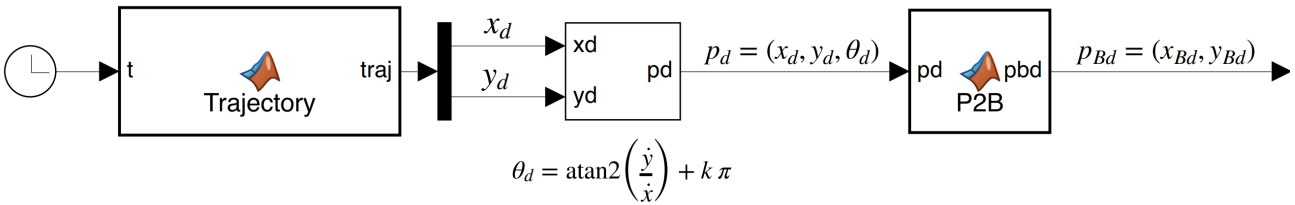
$$\begin{cases} u_1 = K_{P1}\left(x_{Bd} - x_B\right) + \dot{x}_{Bd} \\ u_2 = K_{P2}\left(y_{Bd} - y_B\right) + \dot{y}_{Bd} \end{cases} \qquad K_{P1},\, K_{P2} > 0$$



Trajectory Tracking via Output Feedback: Reference Type

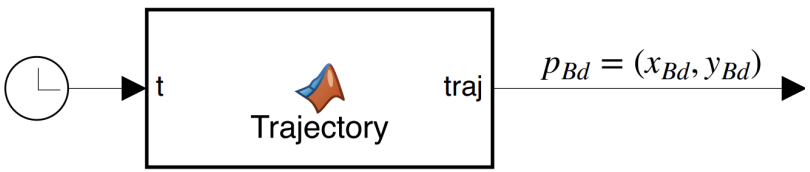
Two different types of reference can be employed for the B-Point controller:

Centroid Reference

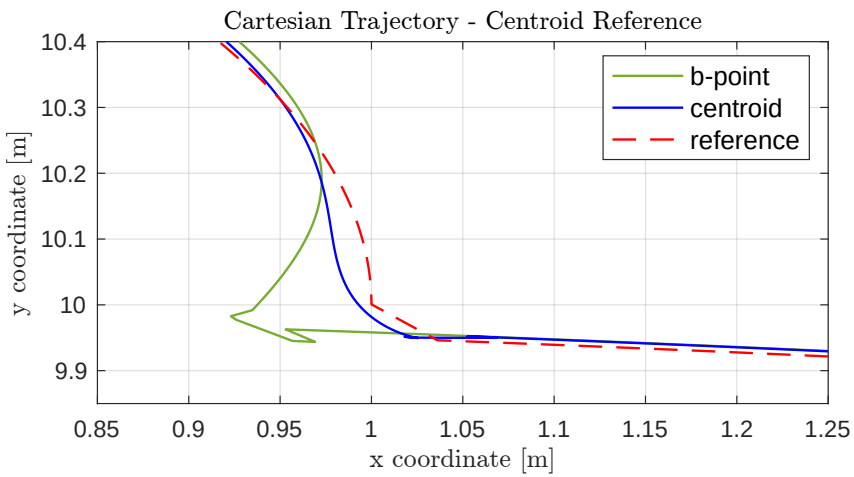


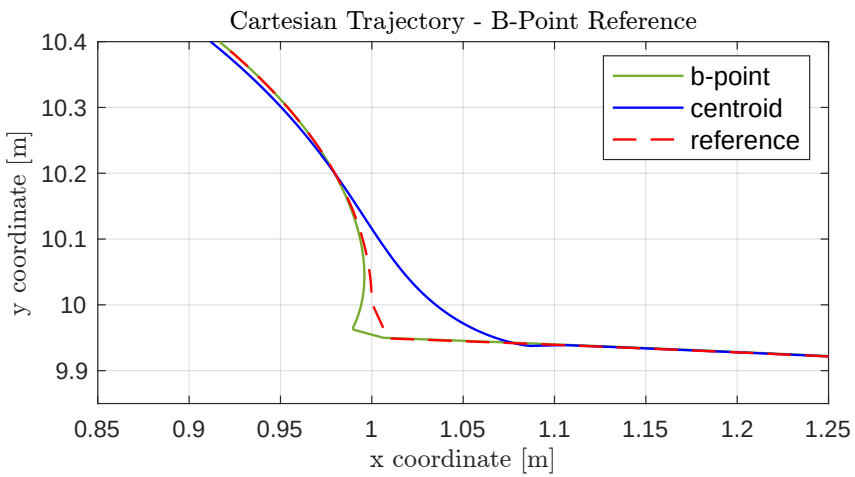
- Allows for more precise tracking
- Subject to numerical errors

B-Point Reference



- Smooth trajectories
- Approximate tracking

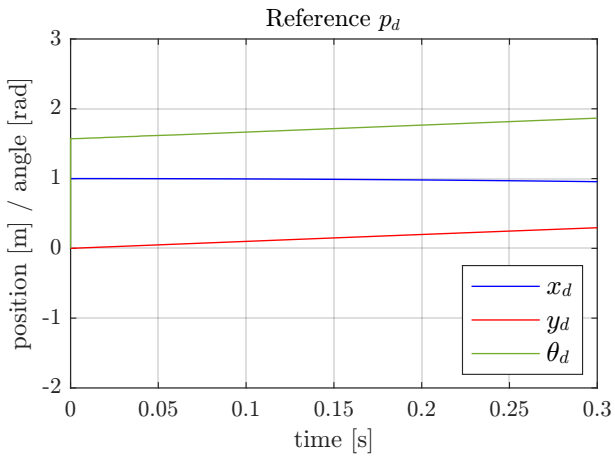


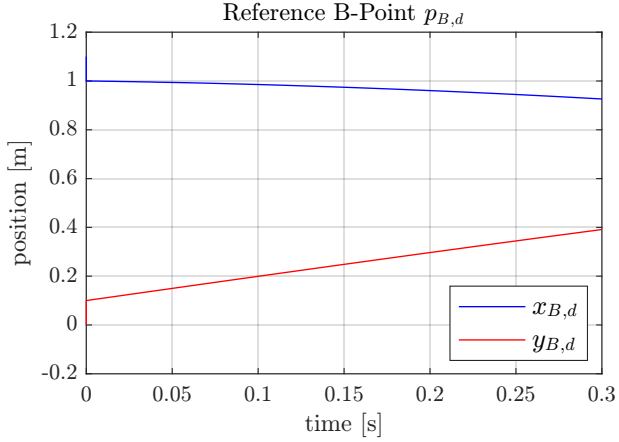


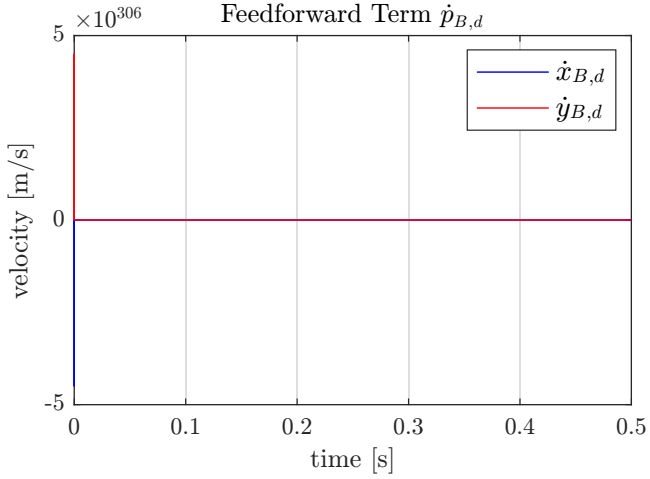
Trajectory Tracking via Output Feedback: Numerical Issues

During the implementation of the B-Point controller, the following numerical error has been encountered:

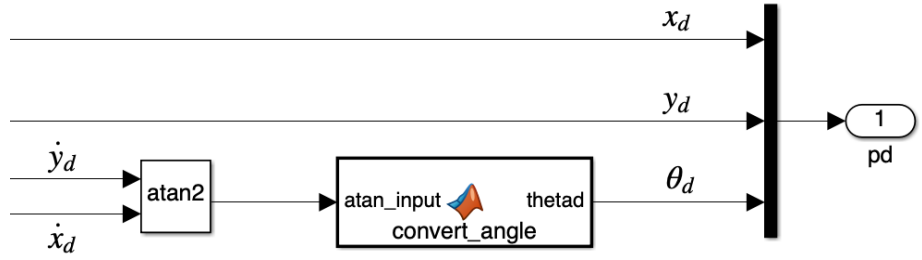
```
An error occurred while running the simulation and the simulation was terminated
Caused by:
  • Derivative of state '1' in block 'unicycle_bpoint_2022b_ISSUE/unicycle_kinematics/Integrator' at time 4.4501477170144E-309
    is not finite. The simulation will be stopped. There may be a singularity in the solution.  If not, try reducing the step
    size (either by reducing the fixed step size or by tightening the error tolerances)
Component: Simulink | Category: Block error
```








```
function thetad = convert_angle(atan_input)
    if(atan_input < 0)
        thetad = atan_input + 2*pi;
    else
        thetad = atan_input;
    end
```



Trajectory Tracking via Output Feedback: Numerical Issues

Possible solutions:

- Saturation block: Ineffective, seems that the issue is not completely related to the magnitude of the derivative term.
- Fixed step-size of the solver: Effective!

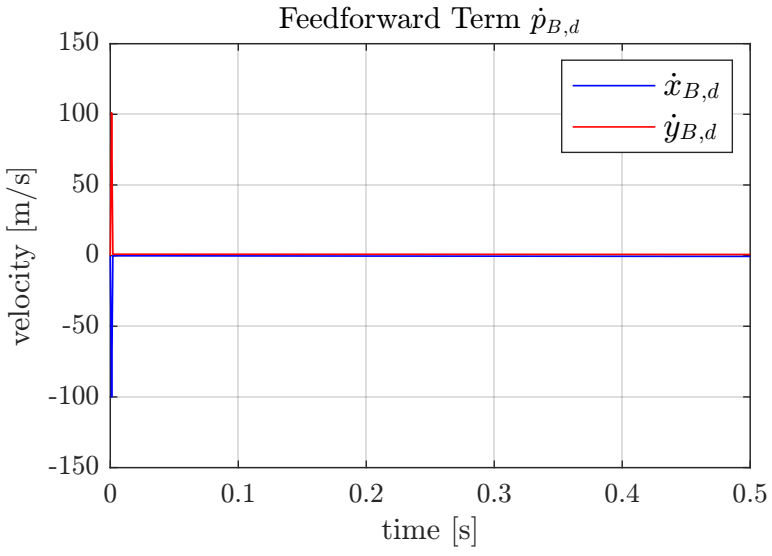


Figure: Step-size $= 10^{-3}$ s

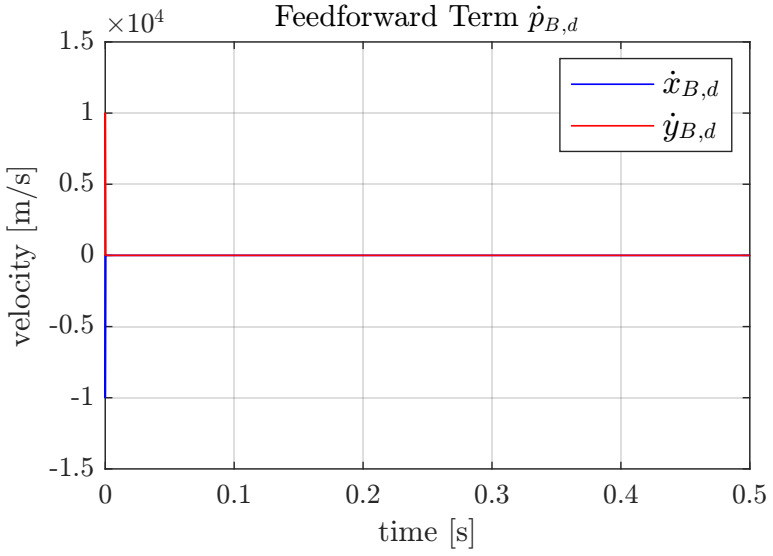
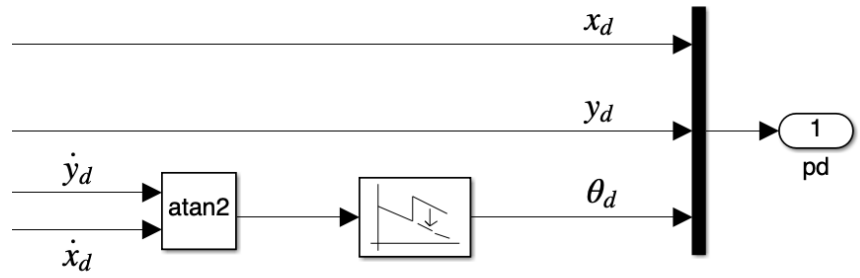
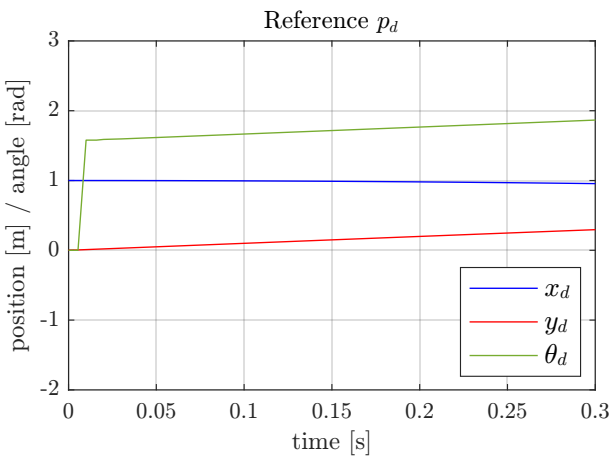


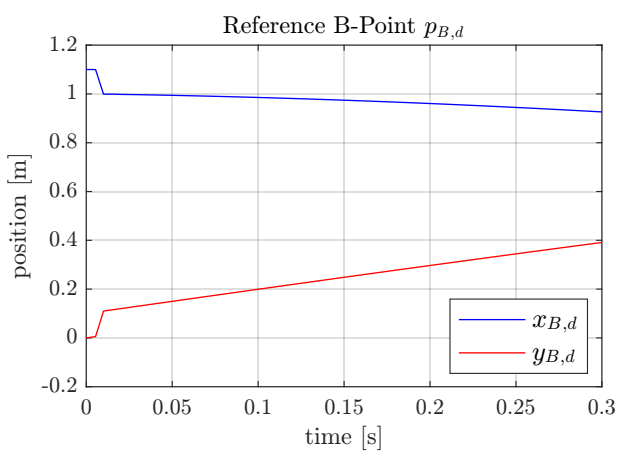
Figure: Step-size $= 10^{-5}$ s

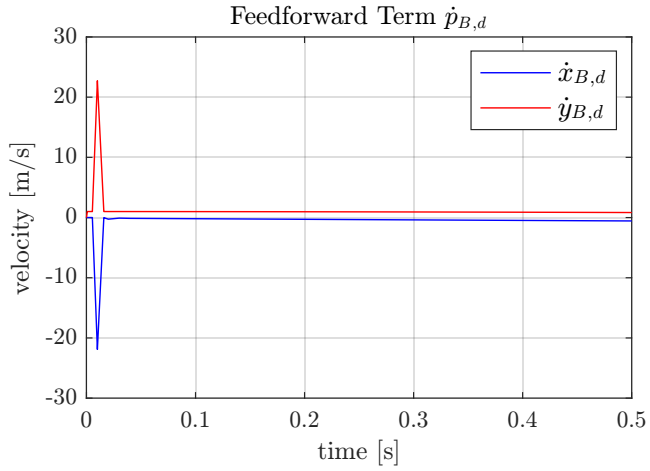
Trajectory Tracking via Output Feedback: Numerical Issues

- Unwrap Block: Effective, since it introduces a delay in the computation of θ_d.



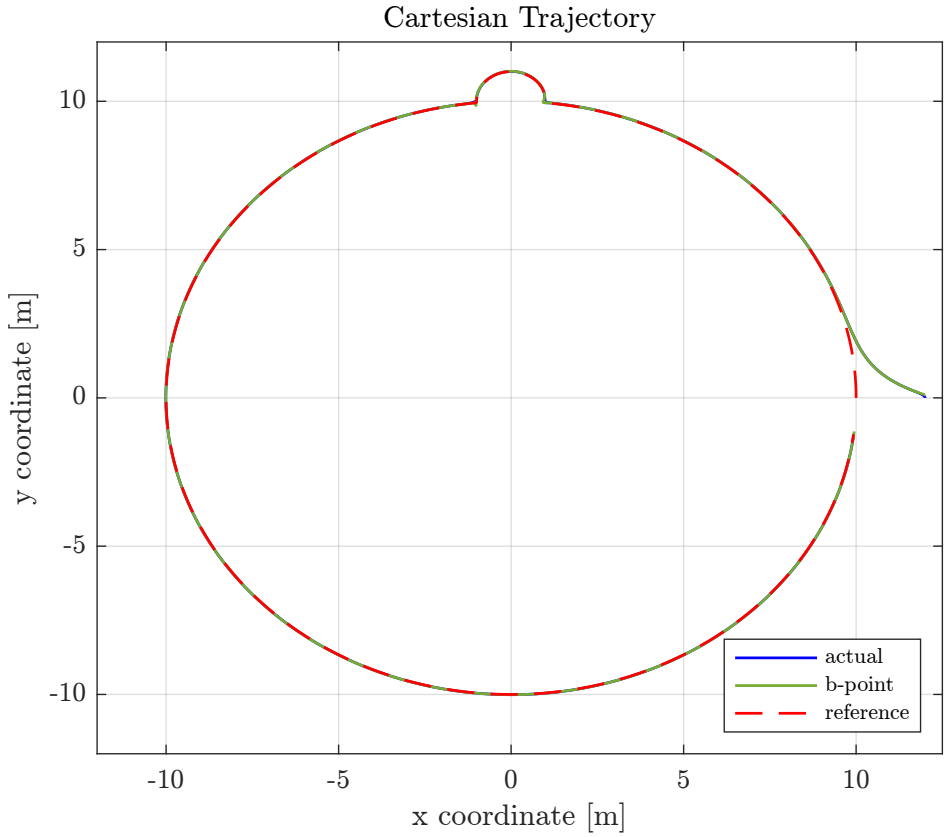


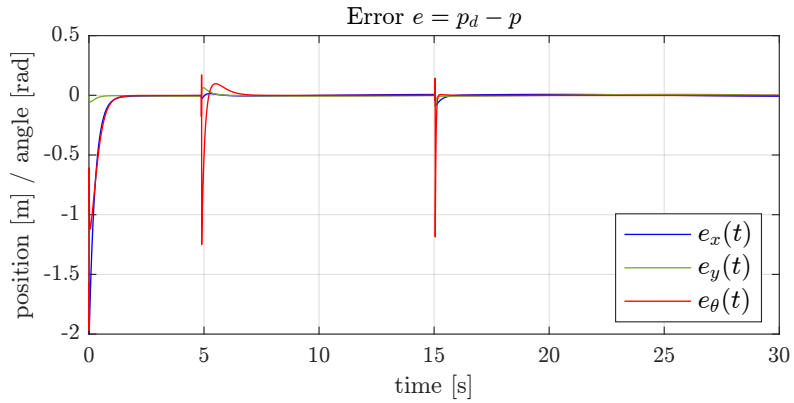


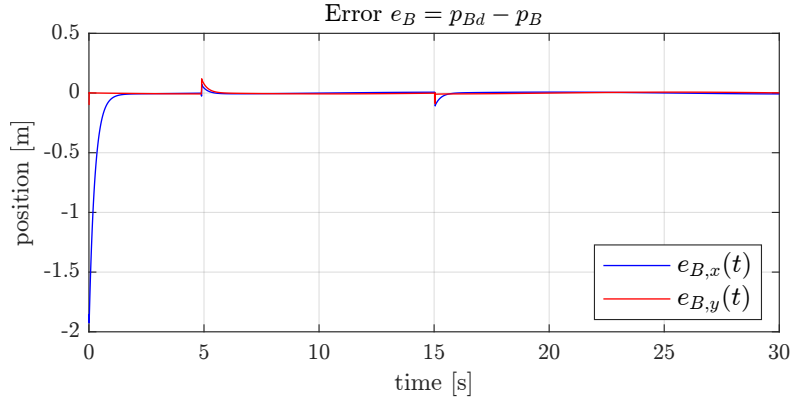


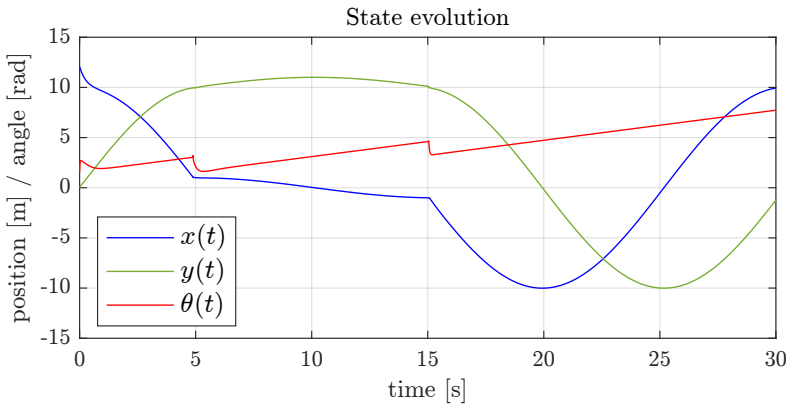
Final observations: try to avoid discontinuities in the reference signal, in particular if it is necessary to compute derivatives of it. If this is not possible, consider adopting pre-computed derivative terms.

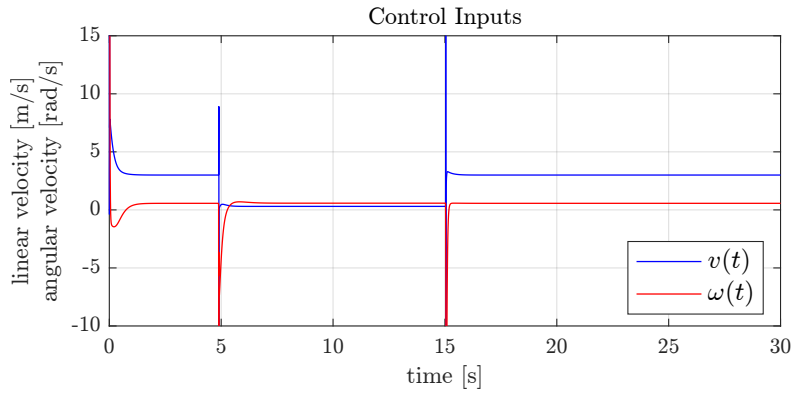
Trajectory Tracking via Output Feedback: Results





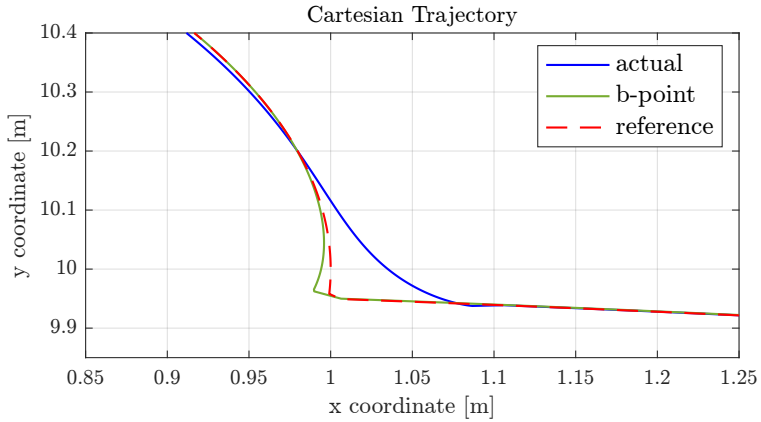


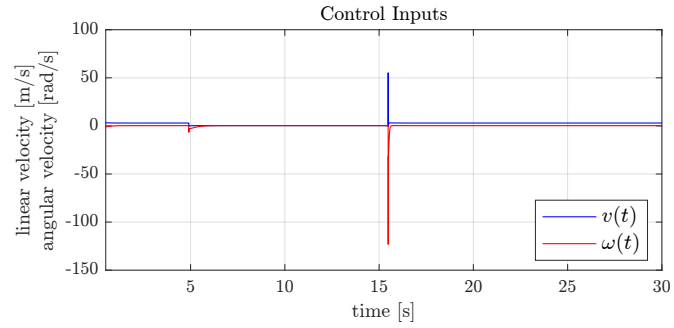




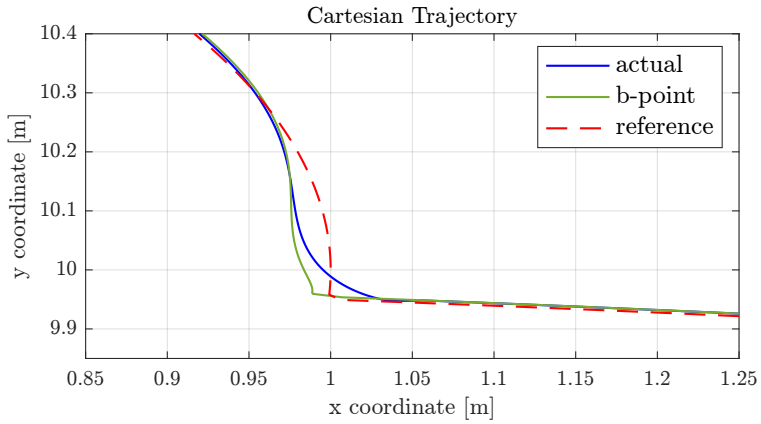
Trajectory Tracking via Output Feedback: Results

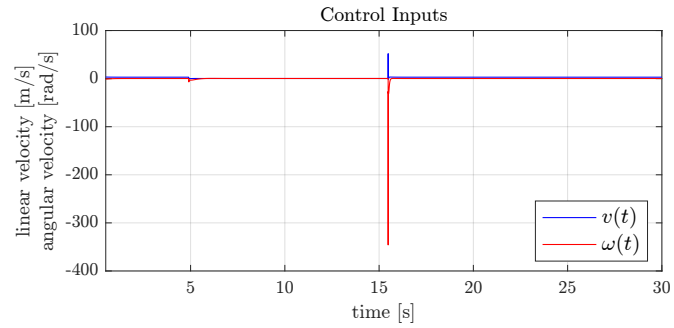
$b = 0.1\,m$:



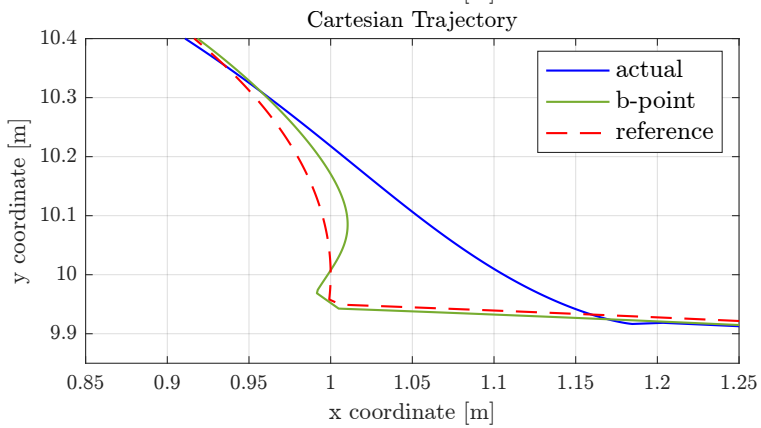


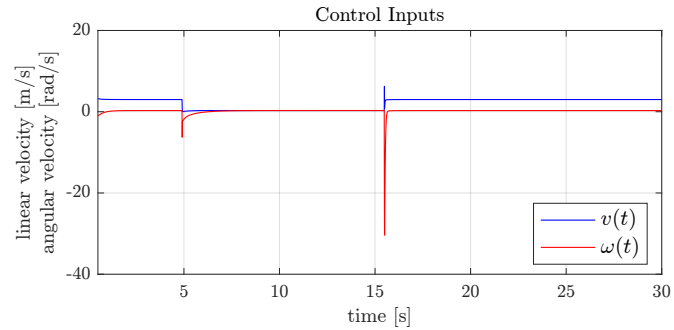
$b = 0.05\,m$:





$b = 0.2\,m$:





Trajectory Tracking: Comparison between State and Output Feedback

	State-Feedback Controller	Output-Feedback Controller
Feedback:	Tracking Error	Cartesian Position Error
	$e_W = p_d - p = \begin{bmatrix} x_d - x \\ y_d - y \\ \theta_d - \theta \end{bmatrix}$	$e_B = p_{Bd} - p_B = \begin{bmatrix} x_{Bd} - x_B \\ y_{Bd} - y_B \end{bmatrix}$
Feedforward:	Differential Flatness	Cartesian Velocities
	$\begin{cases} v = \pm\sqrt{\dot{x}_d^2 + \dot{y}_d^2} \\ \omega = \frac{\dot{x}_d \ddot{y}_d - \dot{y}_d \ddot{x}_d}{\dot{x}_d^2 + \dot{y}_d^2} \end{cases}$	$\dot{p}_{Bd} = \begin{bmatrix} \dot{x}_{Bd} \\ \dot{y}_{Bd} \end{bmatrix}$

Trajectory Tracking: Comparison between State and Output Feedback

	State-Feedback Controller	Output-Feedback Controller
Stability:	A.S. $\iff$ v_d, ω_d constant	A.S. $\iff$ $K_{P1}, K_{P2} > 0$
	$\dot{e} = \begin{bmatrix} -2\xi a & \omega_d & 0 \\ -\omega_d & 0 & v_d \\ 0 & -\frac{a^2-\omega_d^2}{v_d} & -2\xi a \end{bmatrix} e$	$\dot{e}_B = -\begin{bmatrix} K_{P1} & 0 \\ 0 & K_{P2} \end{bmatrix} e_B$
Limitations:	Persistent Trajectory ($v_d \neq 0$) $\implies$ No stopping, No motion inversion	Constraint $b \neq 0$
	$K_2 = \frac{a^2-\omega_d^2}{v_d}$ $\quad a \neq 0$	$\begin{bmatrix} \dot{x}_B \\ \dot{y}_B \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -b\sin(\theta) \\ \sin(\theta) & b\cos(\theta) \end{bmatrix}}_{T(\theta)} \begin{bmatrix} v \\ \omega \end{bmatrix}$
		$\det(T(\theta)) = b$

Table of contents

Constrained trajectory tracking via NMPC

Idea: Formulate the trajectory tracking task in a MPC fashion, so that obstacle avoidance is embedded in the problem.

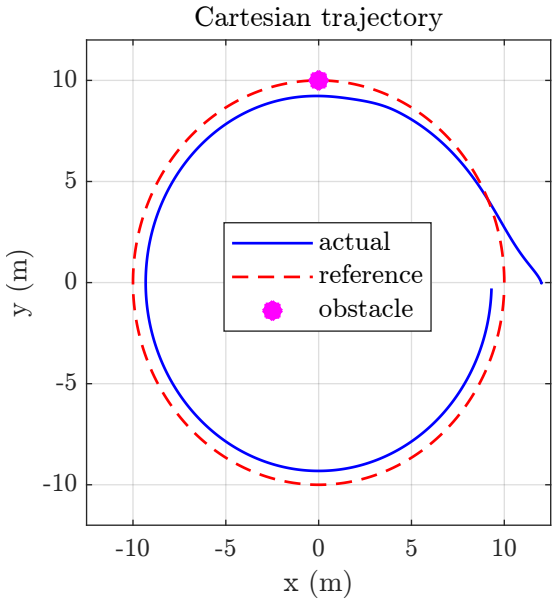
NLP:

$$
\begin{aligned}
\min_{q(\cdot),u(\cdot)} \quad & \int_0^T [\frac{1}{2}||q(t) - ref(t)||^2_{Q_{states}} + \frac{1}{2}||u(t)||^2_{Q_u}]dt + \frac{1}{2}||q(N) - ref(N)||^2_{Q_{N,states}} \\
\text{s.t.} \quad & 0 = q(0) - q_0 \\
& 0 = \dot{q} - f(q(t), u(t)) \quad t \in [0, T] \\
& (x(t) - x_{obs})^2 + (y(t) - y_{obs})^2 > 0.5 \quad t \in [0, T]
\end{aligned} \tag{1}
$$

Tools: Simulations and control design have been carried out using the MATMPC library. In such toolbox, the horizon length is expresses as $T = N \cdot T_{s_{st}}$. Here $T_{s_{st}} = 0.05s$ was used.

Pros & Cons: Obstacle avoidance easy to implement, but the controller is hard to tune and often does not guarantee persistent feasibility.

Constrained trajectory tracking via NMPC - Results



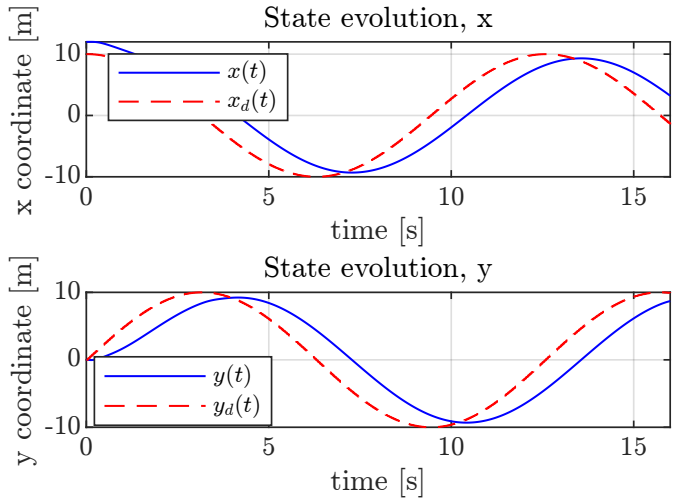
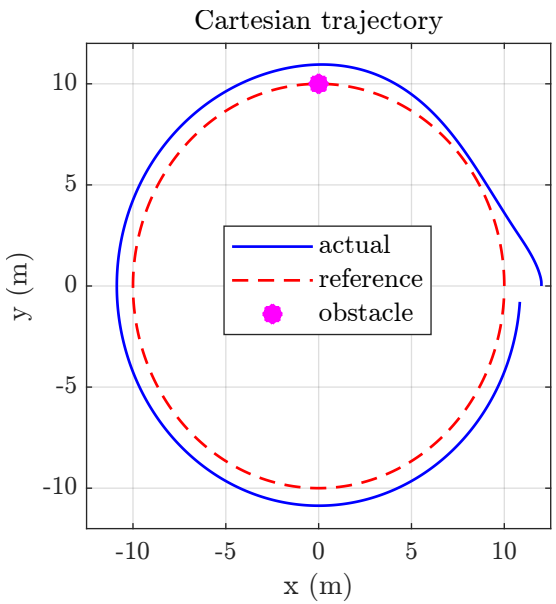


Figure: $N = 100$, $Q = diag\{\underbrace{0.1, 0.1, 0.01}_{Q_{states}}, \underbrace{0.1, 0.1}_{Q_u}\}$



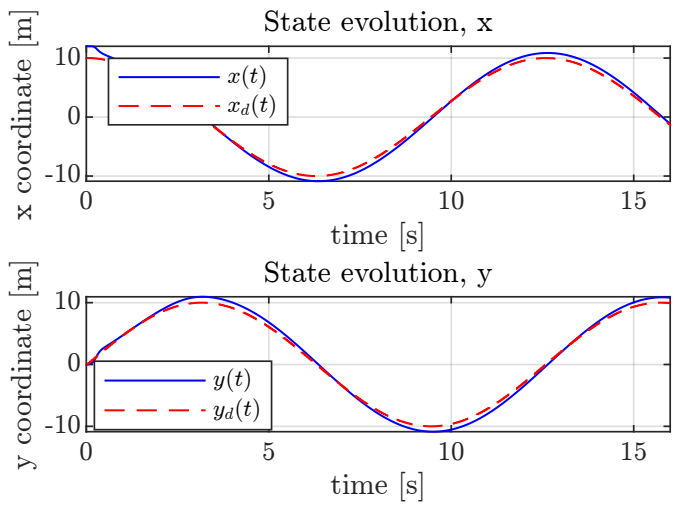


Figure: $N = 5$, $Q = diag\{\underbrace{0.1, 0.1, 0.05}_{Q_{states}}, \underbrace{0.001, 0.01}_{Q_u}\}$

Table of contents

The parking problem

Goal: Once the unicycle completes one lap, it has to park inside a box.

Approaches:

1. Posture regulation
   - Need to specify only the final posture;
   - There is no complete control over the accomplished trajectory.
2. Trajectory tracking
   - Need to design the whole trajectory;
   - Complete control of the maneuver;

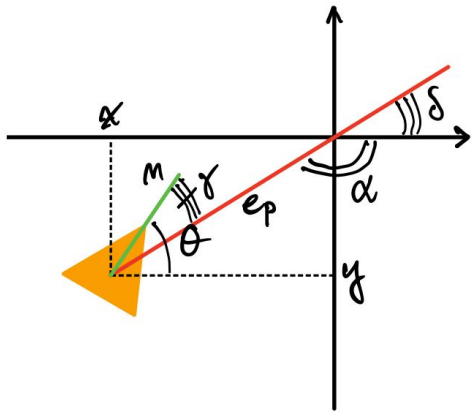
Posture regulation

Goal: From $\bar{q} = [\bar{x}\ \bar{y}\ \bar{\theta}]$ to $q_0 = [0\ 0\ 0]$

Implementation:

1. Convert to polar coordinates

$$\begin{cases} \rho = \sqrt{x^2 + y^2} = |e_\rho| \\ \gamma = atan2(y, x) + \pi - \theta \\ \delta = \gamma + \theta \end{cases}$$



2. Control law

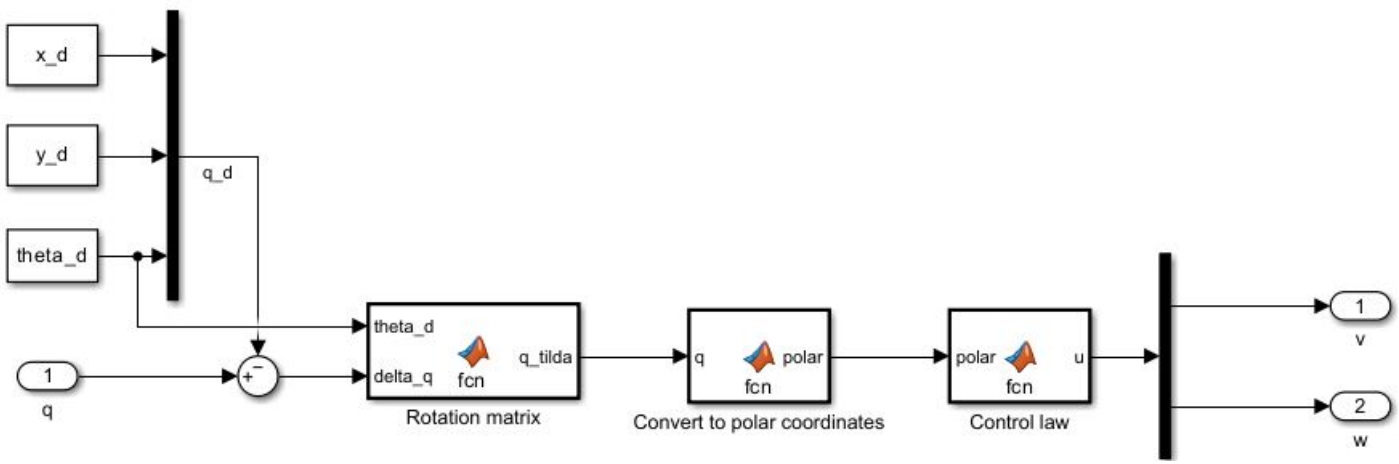
$$\begin{cases} v = K_1\ \rho\ cos(\theta) \\ \omega = K_2\ \gamma + [K_1\ \frac{sin(\gamma)\ cos(\gamma)}{\gamma}\ (\gamma + K_3\ \delta)] \end{cases}$$

Posture regulation - Practical considerations 1

Problem: In practice we want to reach any posture $q_d = [x_d \ y_d \ \theta_d]^T$.
Solution: Apply a change of coordinates before converting to polar ones.

$$\begin{cases} \Delta_q = q - q_d \\ \tilde{q} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta_d) & sin(\theta_d) & 0 \\ -sin(\theta_d) & cos(\theta_d) & 0 \\ 0 & 0 & 1 \end{bmatrix} \Delta_q \end{cases}$$

So, the controller's scheme is



In the next simulations, the target posture is set to $q_d = [12 \quad -2 \quad \frac{\pi}{2}]^T$.

Posture regulation - Practical considerations 2

Problem: Singularity/inputs cause θ to change its steady state after converging to the desired posture.

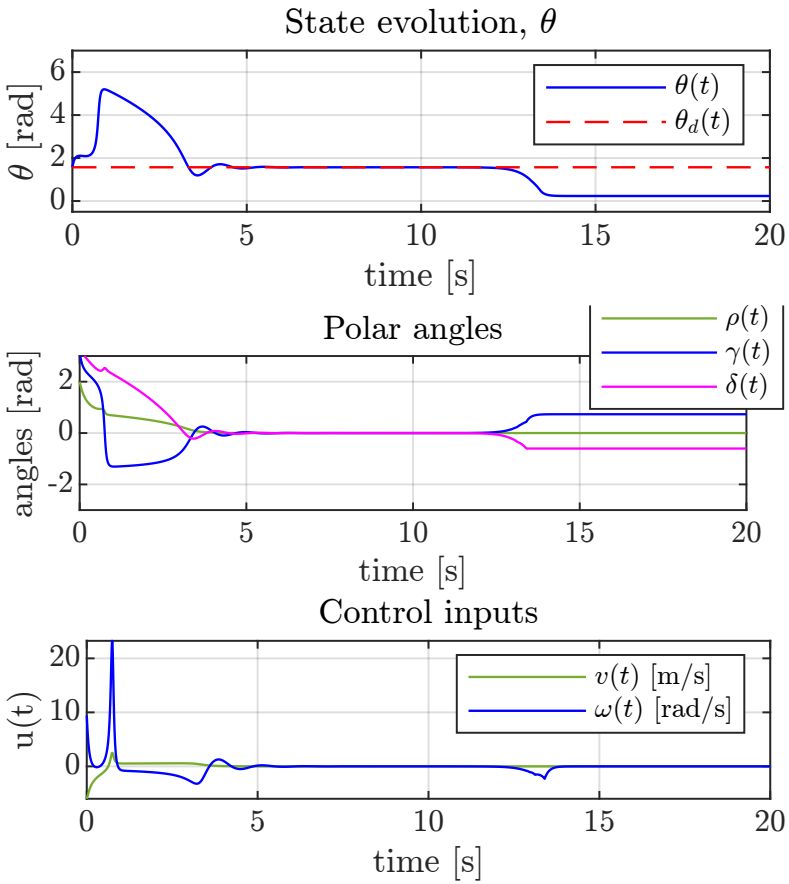


Figure: Controller staying on

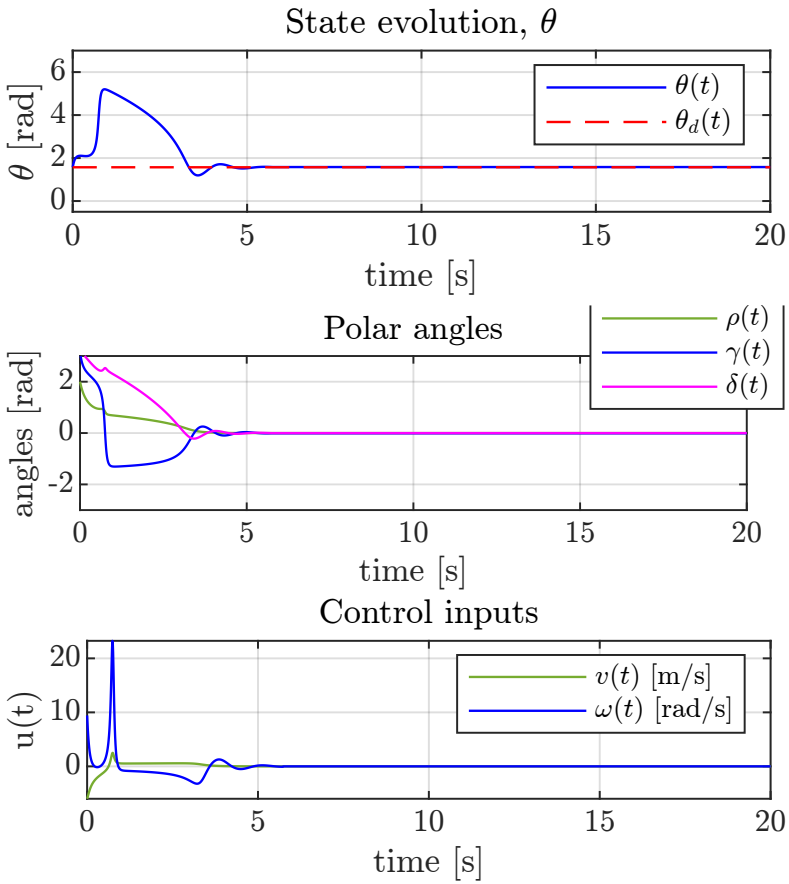
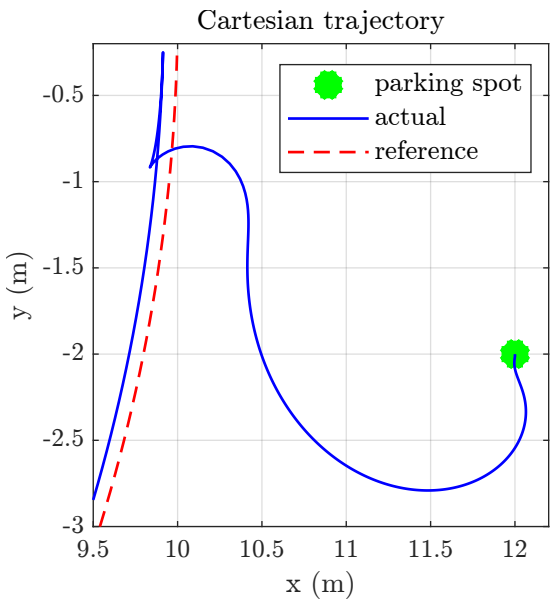


Figure: With our solution

Solution: Stop the controller when ρ, γ and θ are all close to zero.

Posture regulation - Results 1



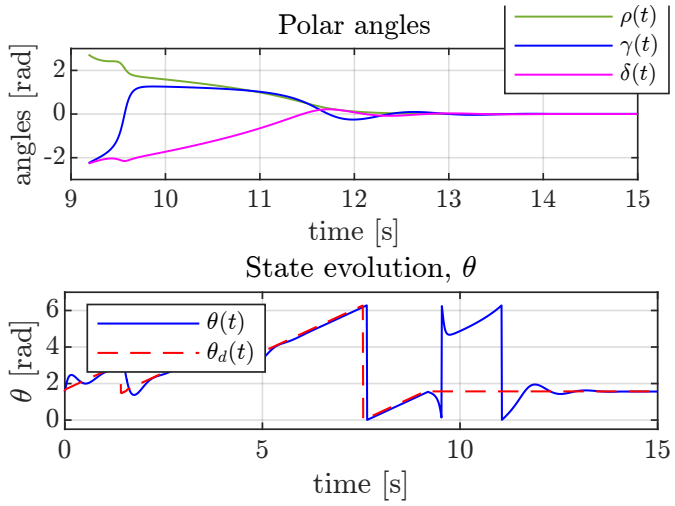
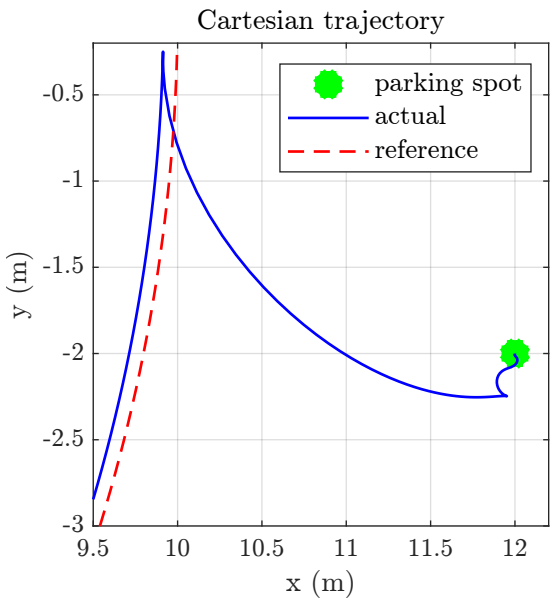


Figure: $K_1 = K_2 = K_3 = 3$



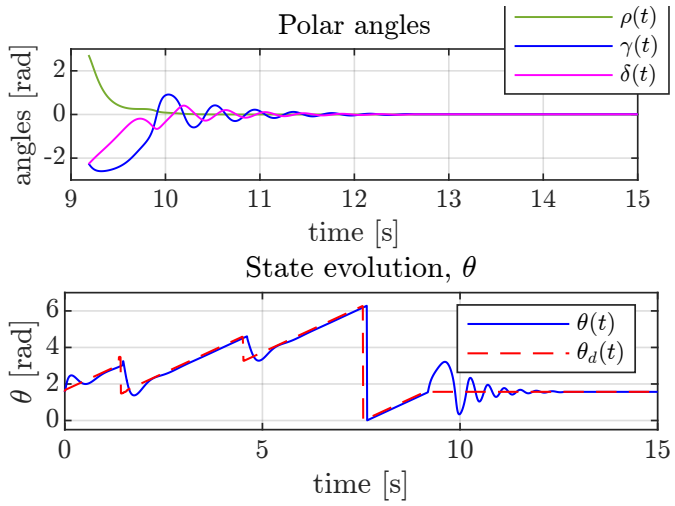
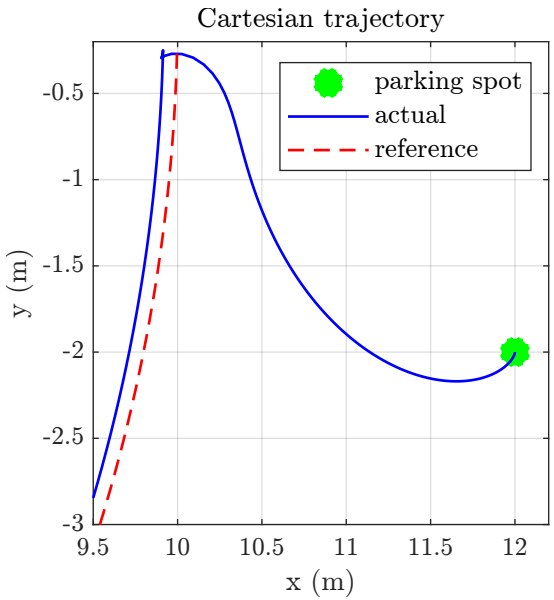


Figure: $K_1 = 9,\ K_2 = K_3 = 3$

Posture regulation - Results 2



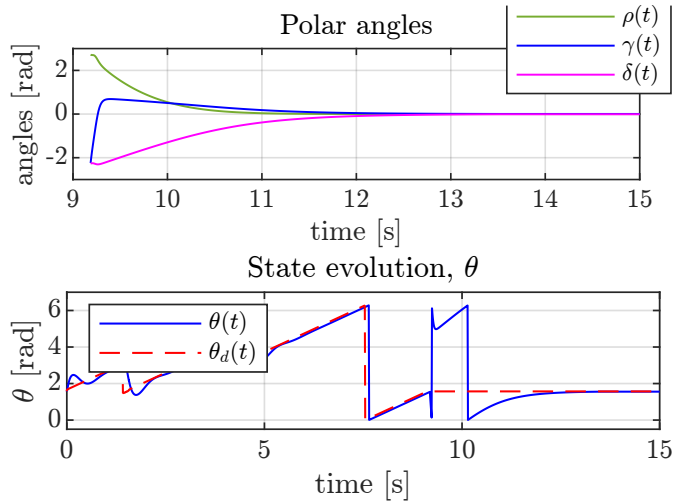
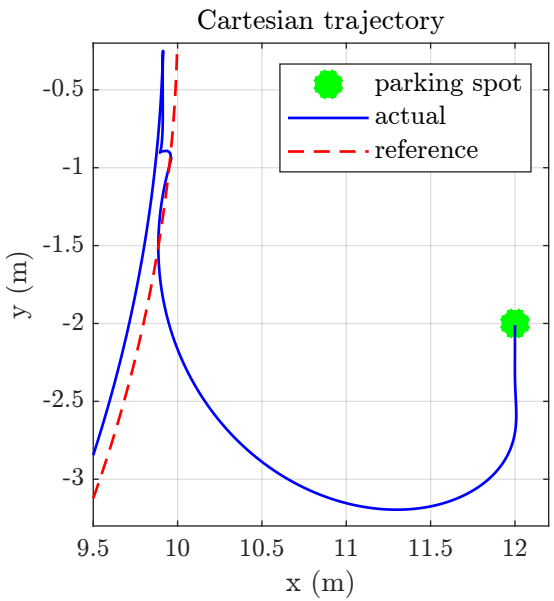


Figure: $K_1 = 3,\ K_2 = 20,\ K_3 = 3$



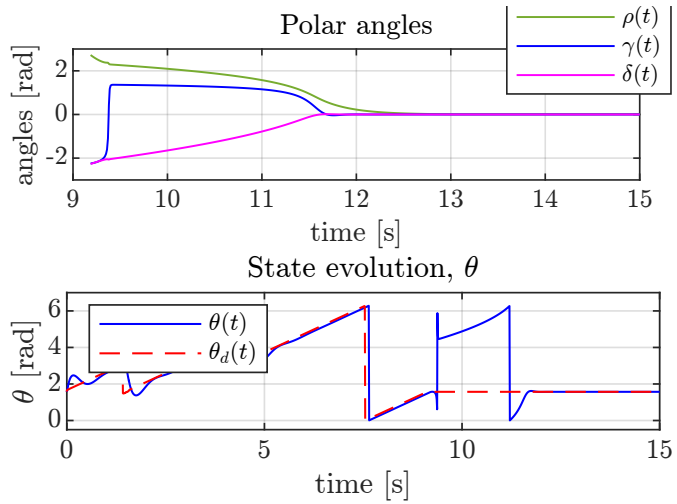


Figure: $K_1 = 3,\ K_2 = 20,\ K_3 = 30$

Full task

Table of contents

Parking maneuver via trajectory tracking

Goal: Design a path that finishes in the target posture $q_d = [14 \quad 0 \quad -\frac{\pi}{2}]^T$ and track it using the previous controllers.

We opted for a half circle trajectory and we compared it with the posture regulator on the same goal.

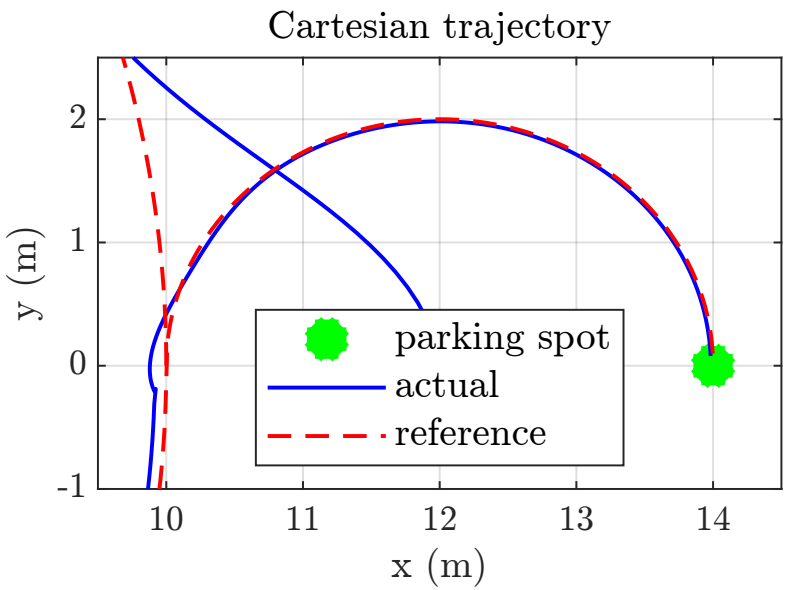


Figure: Full task using trajectory controller

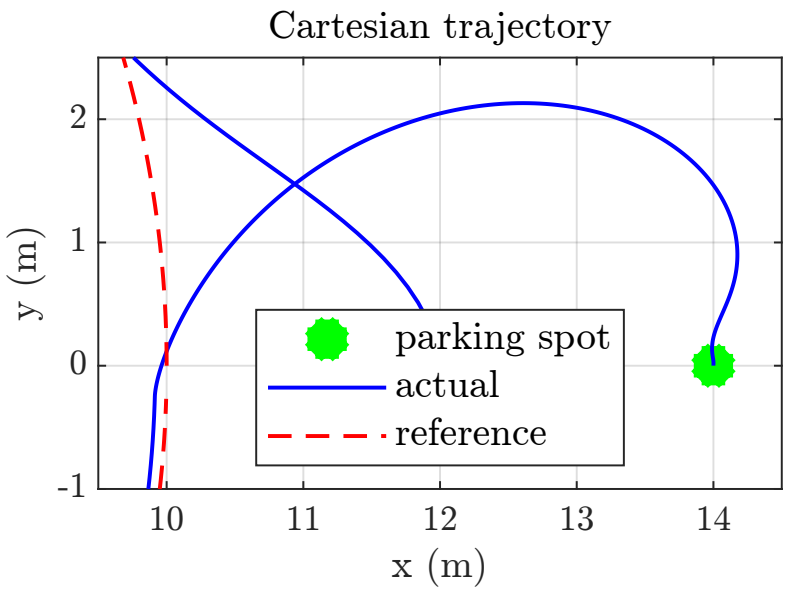


Figure: Full task using posture regulator

Main difficulty is correctly parametrizing the parking trajectory.

Parking maneuver via trajectory tracking - Comparison

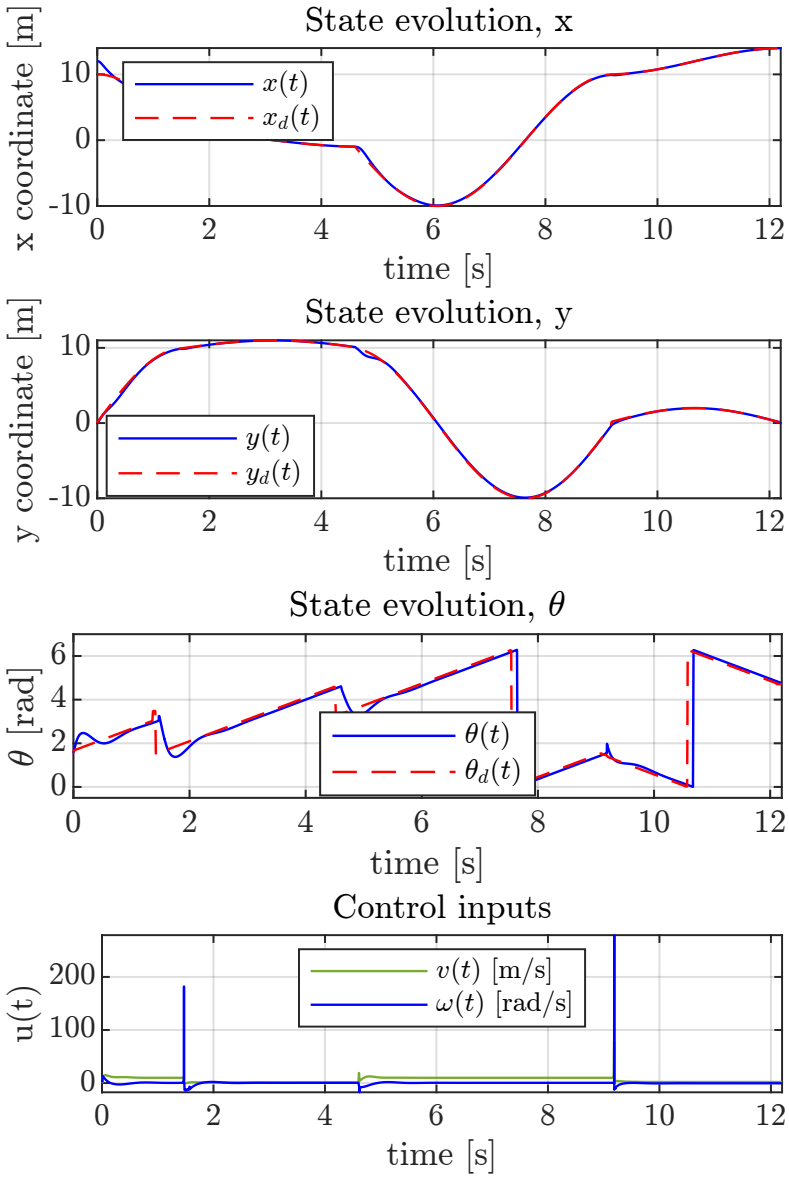


Figure: Trajectory controller

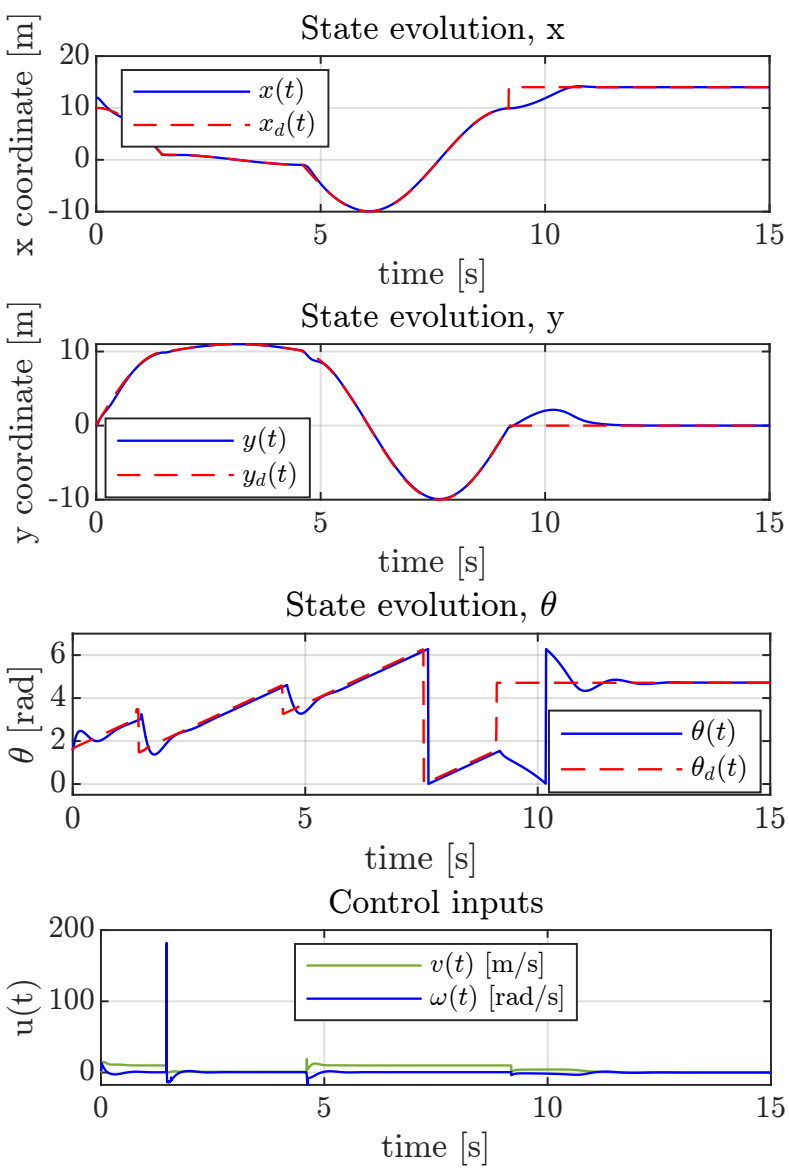


Figure: Posture regulator