Trajectory Tracking Controller Tuning

Control of Mobile Robots

Alessio Onori

January 31, 2025

1 Introduction

This document describes how I tuned a trajectory tracking controller for a differential-drive robot based on an extended unicycle kinematic model. The controller consists of:

- A PI (Proportional-Integral) loop in both the x and y channels.
- Feedback linearization with a tunable lookahead distance, P_{distance}.
- A sampling interval T_s that is chosen to balance theoretical bandwidth requirements.

The main objective was to minimize the tracking errors in x and y while maintaining a desired phase margin and limiting overshoot. My tuning strategy combined:

1. **Control Theory (Bode Analysis)** to derive initial gain values.
2. **Simulation and Empirical Testing** to validate the performance and refine the parameters based on observed tracking and robustness.

2 First Attempt: Theoretical Gains (8,5)

I began with classical Bode analysis to obtain a phase margin greater than 45° and an overshoot around 20%. This step yielded:

$$K_{\text{p},x} = 8.0, \quad T_{\text{i},x} = 5.0$$

(with similar values for the y-channel). According to open- and closed-loop frequency responses, these gains gave:

- Phase margin $\approx 46.26^\circ$ at $\omega = 5.97\,\text{rad/s}$.
- Closed-loop rise time $\approx 0.21\,\text{s}$.
- Closed-loop settling time $\approx 1.74\,\text{s}$.
- Overshoot $\approx 23.62\%$.
- Bandwidth $\approx 9.705\,\text{rad/s}$.

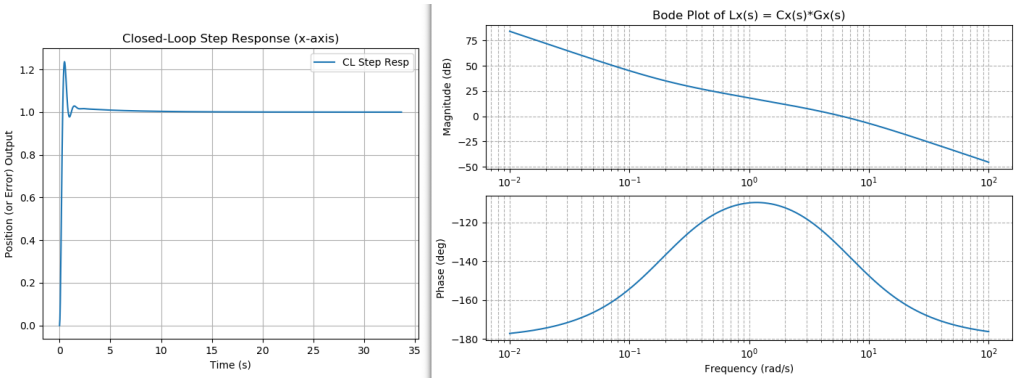

Figure 1: Bode and Closed Loop

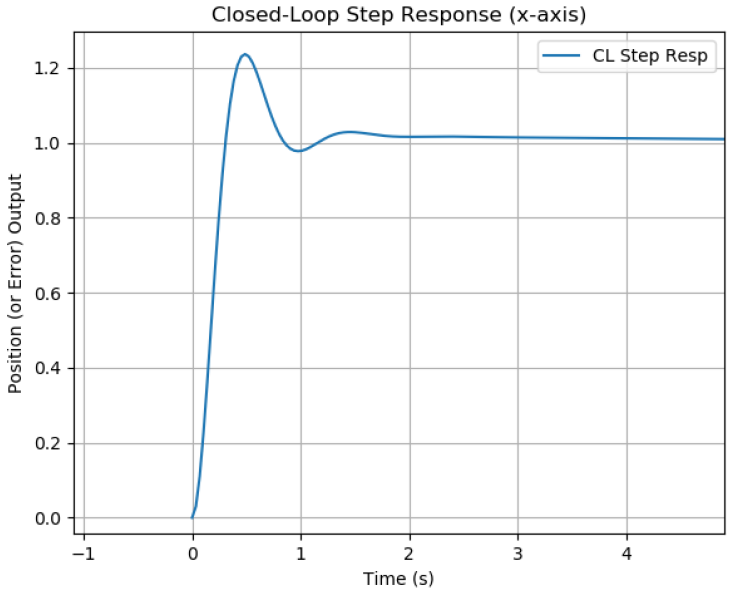

Figure 2: Closed Loop focus

Practical Testing and Unexpected Issues

Despite looking good in theory, this configuration performed poorly in real (or simulated) tests. The robot visually failed to follow the desired path closely. At this time, I did not realize that the *lookahead distance* parameter, P_{distance}, was interfering with the effectiveness of these gains.

3 Second Attempt: Relaxed Requirements (13,4)

In an effort to improve tracking, I decided to *relax my phase margin requirement*. That is, I allowed lower margins if it meant faster response and better path convergence. A second round of Bode analysis suggested:

$$K_{\text{p},x} = 13.0, \quad T_{\text{i},x} = 4.0$$

which produced:

- Phase margin $\approx 37.36^\circ$ at $\omega = 8.20\,\text{rad/s}$.
- Closed-loop rise time $\approx 0.16\,\text{s}$.
- Closed-loop settling time $\approx 1.28\,\text{s}$.
- Overshoot $\approx 33.48\%$.
- Bandwidth $\approx 13.278\,\text{rad/s}$.

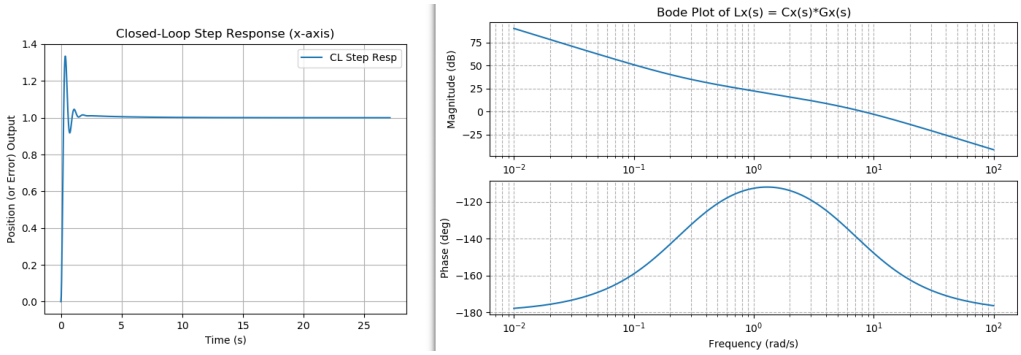

Figure 3: Bode and Closed Loop

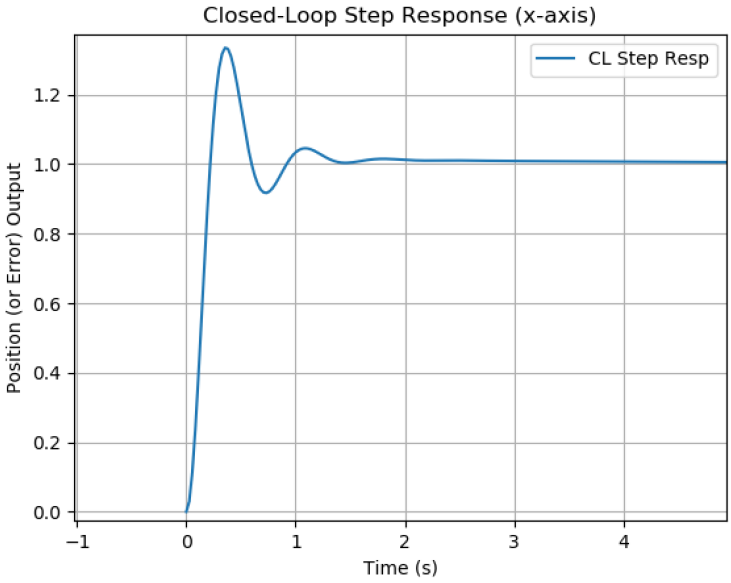

Figure 4: Closed Loop focus

Practical Testing

Empirically, these gains performed *better* than $(8, 5)$ under the same conditions. The tracking was visually improved, but I still was not satisfied: the shape of the path was not similar enough compared to the reference.

4 Crucial Realization: Pdistance Matters

Eventually, I discovered that the *lookahead distance* P_{distance} was playing a dominant role. Originally, I had set

$$P_{\text{distance}} = 2.0\,\text{m},$$

which placed the control point far ahead of the robot's center, causing slower lateral corrections and contributing to large overshoot in tight maneuvers.

Reducing Pdistance

I drastically lowered this parameter to

$$P_{\text{distance}} = 0.05\,\text{m}.$$

With the control point much closer to the robot's center, both sets of gains $(8, 5)$ and $(13, 4)$ performed significantly better. In particular:

- **Lateral corrections** happened more quickly.
- **Overshoot and phase margin** were easier to balance.
- The **tracking trajectory** was much smoother and more accurate.
- **Velocity Commands**: Both linear and angular velocity commands remained within reasonable ranges, preventing excessive stress on the actuators. The initial (maximum) rapid movements were not too steep, ensuring smoother operation and longevity of the motors.

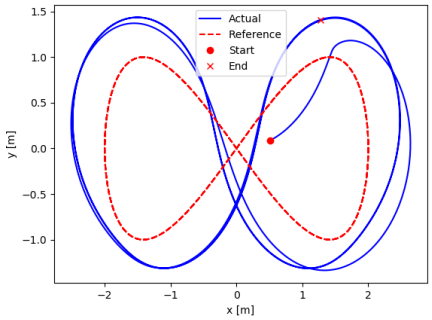

Figure 5: Effect of Large P_{distance} (13,4)

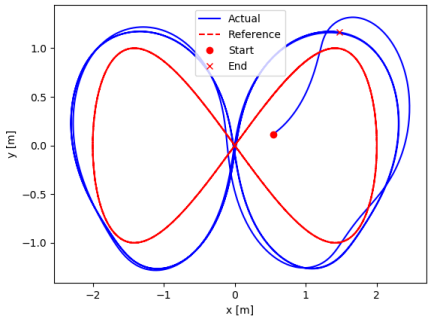

Figure 6: Effect of Large P_{distance} (8,5)

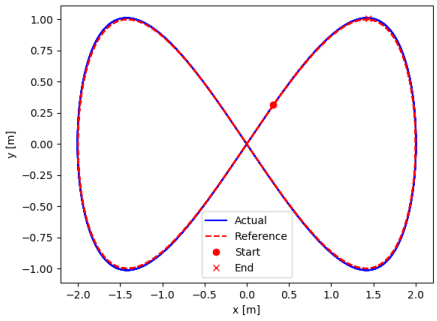

Figure 7: Effect of Low P_{distance} (8,5)

5 Final Decision: Return to (8,5)

Given the improved performance with a smaller P_{distance}, I revisited my original theoretical gains of $(8, 5)$. Thanks to the shorter lookahead distance, these gains now provided:

- A **safer phase margin** of around 46°.
- **Lower overshoot** (closer to my target of $\sim 20\%$).
- A **smooth tracking response** that looked good in simulation and practice.
- **Reasonable Velocity Commands**: The linear and angular velocity commands were within expected operational ranges, ensuring that the robot's movements were smooth and that actuators were not subjected to undue stress from abrupt commands.

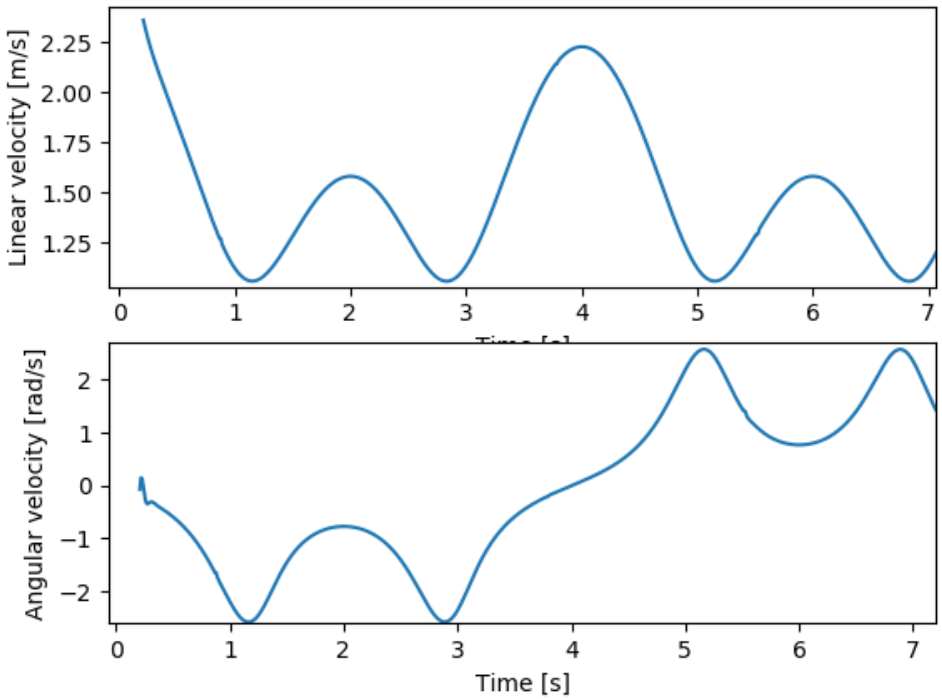

Figure 8: Reasonable Velocity commands

As a result, I settled on:

$$K_{\mathrm{p},x} = 8, \quad T_{\mathrm{i},x} = 5, \quad K_{\mathrm{p},y} = 8, \quad T_{\mathrm{i},y} = 5, \quad P_{\mathrm{distance}} = 0.05\,\mathrm{m}.$$

Note that I decided to keep $K_{\mathrm{p,x}} = K_{\mathrm{p,y}}$ and $T_{\mathrm{i,x}} = T_{\mathrm{i,y}}$ since the performances were already quite satisfactory.

6 Sampling Time Ts

From the closed-loop bandwidth estimates (roughly 9.7 rad/s to 13.3 rad/s), a common rule of thumb $\left(T_s \leq \frac{1}{10 f_{\mathrm{BW}}}\right)$ suggested $T_s \approx 0.05\,\mathrm{s}$. However, in practice, I chose:

$$T_s = 0.01\,\mathrm{s}$$

for three reasons:

1. It was more **responsive and stable** in the simulator, mitigating lag issues around tight turns, and initial oscillations also.
2. Higher-frequency control updates (100 Hz) did not overload the system, and data logging was still manageable in ROS bags at this rate.
3. Higher-frequency simulation updates (dt integration step < 0.01 s) did make the rosbag recording to struggle on my hardware. 100Hz was therefore the lower bound, which matched perfectly with the control update frequency requirement.

I also verified the effects of running the control update at a lower frequency:

- $T_s = 0.03\,\mathrm{s} - 0.05\,\mathrm{s}$: Resulted in noticeable initial oscillations.
- $T_s > 0.05\,\mathrm{s}$: Led to poor performance and inadequate trajectory tracking.

7 How to Run the code

Each of the two packages have a README.md file that can help in running the simulation, the analysis, and the plotting. The implementation has been tested on Ros1 Noetic, as per requirement. A Dockerfile is provided mainly to show exactly the few dependencies that are needed.

Preview README.md

the simulator takes the desired velocity commands (linear and angular) and integrates the unicycle model to produce the next state.
In order to simulate the motors and the low-level velocity controllers, the unicycle kinematic model is extended with transfer functions.

PARAMS PersonCode a T Ta 10439310 2.0 8.0 0.150

Requirements

- as per instruction: `projects are verified using ROS Noetic`
- Check Dockerfile to see exactly what packages are required (Included here in the package src just for this purpose)

HOW TO RUN PART 1, with simple test node

```
catkin_make
roslaunch turtlebot_simulator turtlebot_simulator_test_with_bag.launch # a bag will be recored for 25 (sim_time) seconds -> please stop manually with ctrl+c
# roslaunch turtlebot_simulator turtlebot_simulator_test.launch # if you don't want the bag recording
python3 script/plot_result.py trajectory.bag
```

README of the simulator

Preview README.md

Modules: Trajectory Tracking Controller + Feedback Linearizer Controller with point P which is ditant P_dist. PI is embedded in Trajectory Tracking Controller module

PARAMS

PersonCode a T Ta
10439310 2.0 8.0 0.150

Requirements

- as per instruction: `projects are verified using ROS Noetic`
- Check Dockerfile to see exactly what packages are required (Included here in the package src just for this purpose)
  - i.e. to run the python scripts: `pip3 install matplotlib control numpy`

HOW TO RUN PART 2 (SIMULATOR+CONTROLLER)

```
catkin_make
roslaunch turtlebot_traj_ctrl turtlebot_traj_ctrl_with_bag.launch # a bag will be recored for 25 (sim_time) seconds -> please stop manually with ctrl+c
# roslaunch turtlebot_traj_ctrl turtlebot_traj_ctrl.launch # if you don't want the bag recording
python3 script/plot_result.py trajectory.bag
```

HOW TO RUN BODE ANALYSIS

```
python3 script/bode_tuning_x.py
# Tune Kp_x and T1_x in the code
```

README of the controller

turtlebot_simulator.yaml

catkin_ws > src > turtlebot_simulator > config > turtlebot_simulator.yaml

```yaml
simulator:
  num_steps: 300000 #more than enough to record 25 seconds of simulation
  sim_speed: 1
  dt: 0.01 #integration step
  Ta: 0.150
  initial_conditions:
    x0: 0.0
    y0: 0.0
    theta0: 0.0

#useful only for the assignment number 1
test:
  run_period: 0.01
  T01: 2
  V0: 0.0
  V1: 0.5
  omega0: 0.0
  omega1: 2.0
```

controller_params.yaml

catkin_ws > src > turtlebot_traj_ctrl > config > controller_params.yaml

```yaml
controller:
  Ts: 0.01 #sampling time
  P_dist: 0.05

trajectory_controller:
  T: 8.0
  alfa: 2.0

  K_prop_x: 8.0
  K_prop_y: 8.0
  T_integral_x: 5.0
  T_integral_y: 5.0

  # K_prop_x: 13.0
  # K_prop_y: 13.0
  # T_integral_x: 4.0
  # T_integral_y: 4.0
```

The 2 config files

8 Conclusion, Figures and Results

My final trajectory tracking controller parameters are:

$$K_{\mathrm{p},x} = 8, \quad T_{\mathrm{i},x} = 5, \quad K_{\mathrm{p},y} = 8, \quad T_{\mathrm{i},y} = 5,$$
$$P_{\mathrm{distance}} = 0.05\,\mathrm{m}, \quad T_s = 0.01\,\mathrm{s}$$

The given fixed parameters given my person code are:

$$a = 2.0, \quad T = 8.0 \quad T_a = 0.150$$

By reducing P_{distance}, I enabled the original "safer" theoretical gains $(8, 5)$ to achieve better real-world tracking than the more aggressive $(13, 4)$ set. The smaller lookahead distance helped maintain a higher phase margin and reduced overshoot, while a 0.01 s sampling time ensured responsiveness without overburdening the simulator or data logging. Additionally, with this configuration, both linear and angular velocity commands remained within reasonable ranges, and the initial (maximum) rapid movements were not too steep, thereby preventing excessive stress on the actuators and ensuring smoother and more reliable robot operation.

The final maximum tracking errors are approximately 0.015 m along the x-axis and 0.01 m along the y-axis, which is also evident in the visually accurate trajectory. These results confirm that the tuning process has successfully met the desired specifications. Further investigations could explore alternative techniques or more challenging initial conditions, though such studies fall outside the scope of the current analysis.

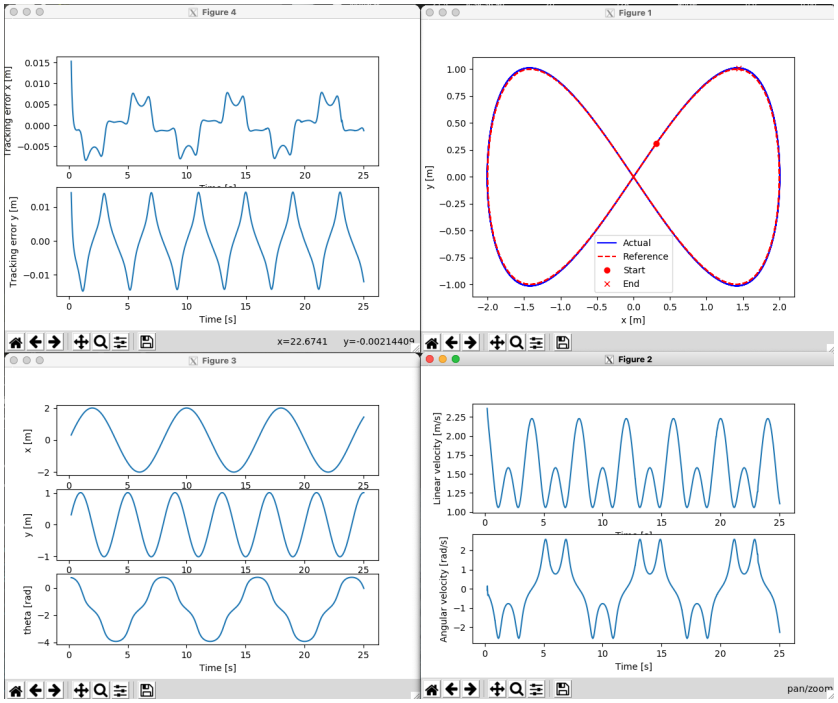

Figure 9: Overall result

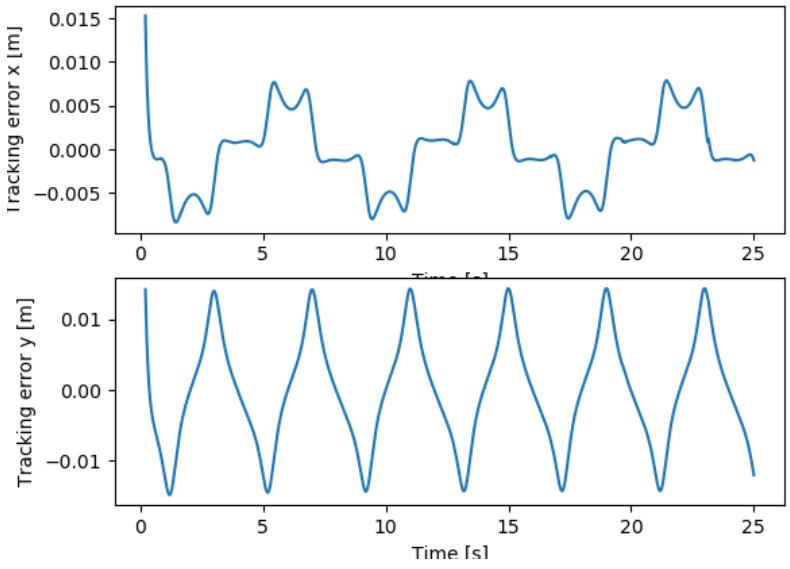

Figure 10: Tracking error

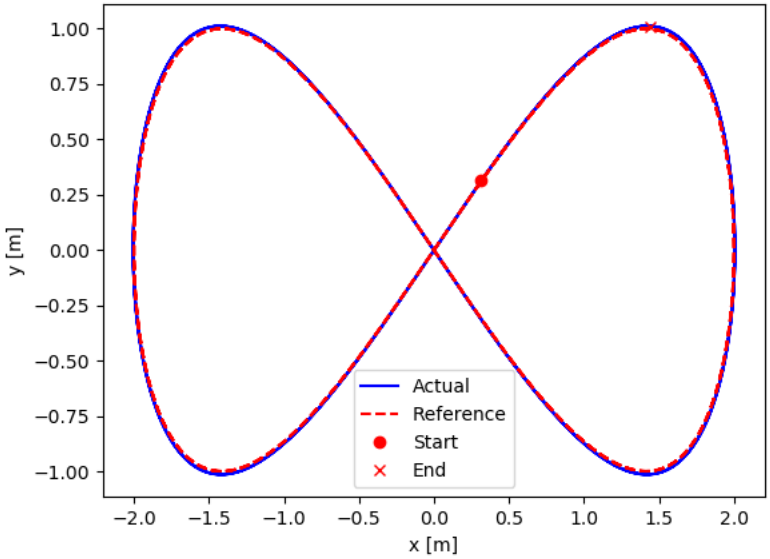

Figure 11: Trajectory

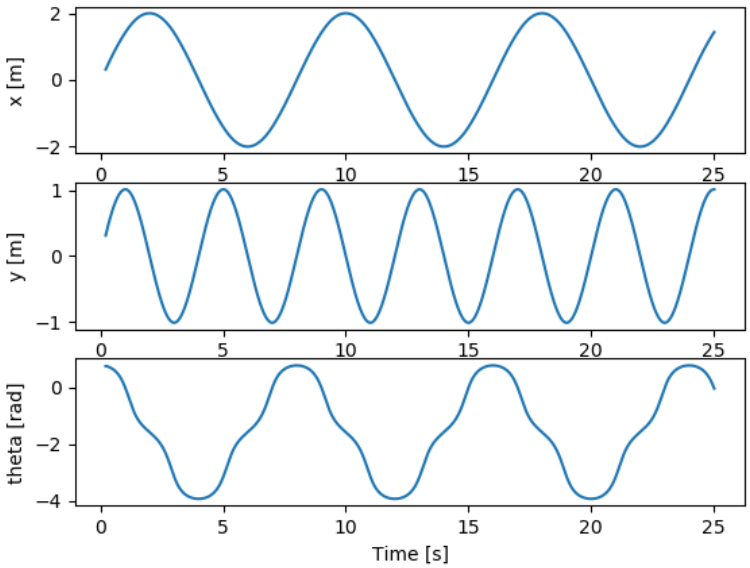

Figure 12: State

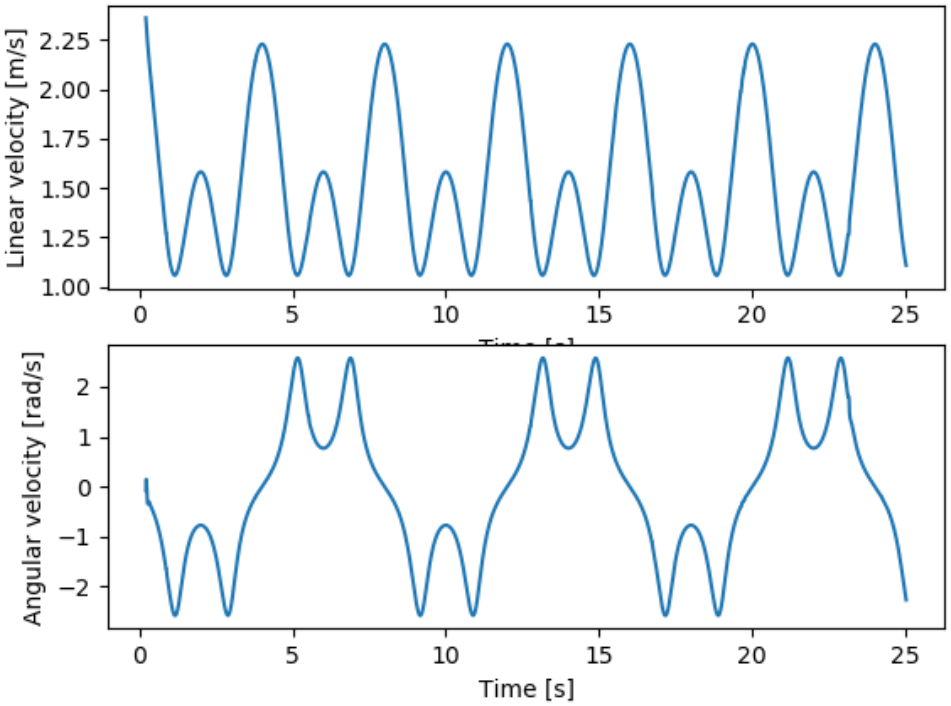

Figure 13: Control signals