



ARTIFICIAL INTELLIGENCE AND ROBOTICS

LM IN ARTIFICIAL INTELLIGENCE

Assignment 1

SUBMISSION 1

Students :

Álvaro ESTEBAN MUÑOZ

Teacher :

Alessandro SAFFIOTTI

Andrea GOVONI

March 26, 2024

Contents

1	Implement the navigation loop in ROS	2
1.1	The problem	2
1.2	Objects and Data Structures	2
1.3	Control and Information flow	3
1.4	Parameters	3
1.5	Results Obtained	4
1.6	Discussion	6
2	Test the quality of your position estimation more extensively	6
2.1	The problem	6
2.2	Objects and Data Structures	6
2.3	Results Obtained	7
2.4	Discussion	9
3	Implement a linear control strategy to navigate to a given goal position	9
3.1	The Problem	9
3.2	Parameters	9
3.3	Results Obtained	10
3.4	Discussion	11

1 Implement the navigation loop in ROS

1.1 The problem

In this first task we are asked to implement the different parts of the **Robot Control Program** (RCP). These parts are the "Read Sensors", "Estimate State", "Decide Action" and "Send Action" boxes that can be seen on figure [1]. Then, a qualitative test of the implemented program has to be performed.

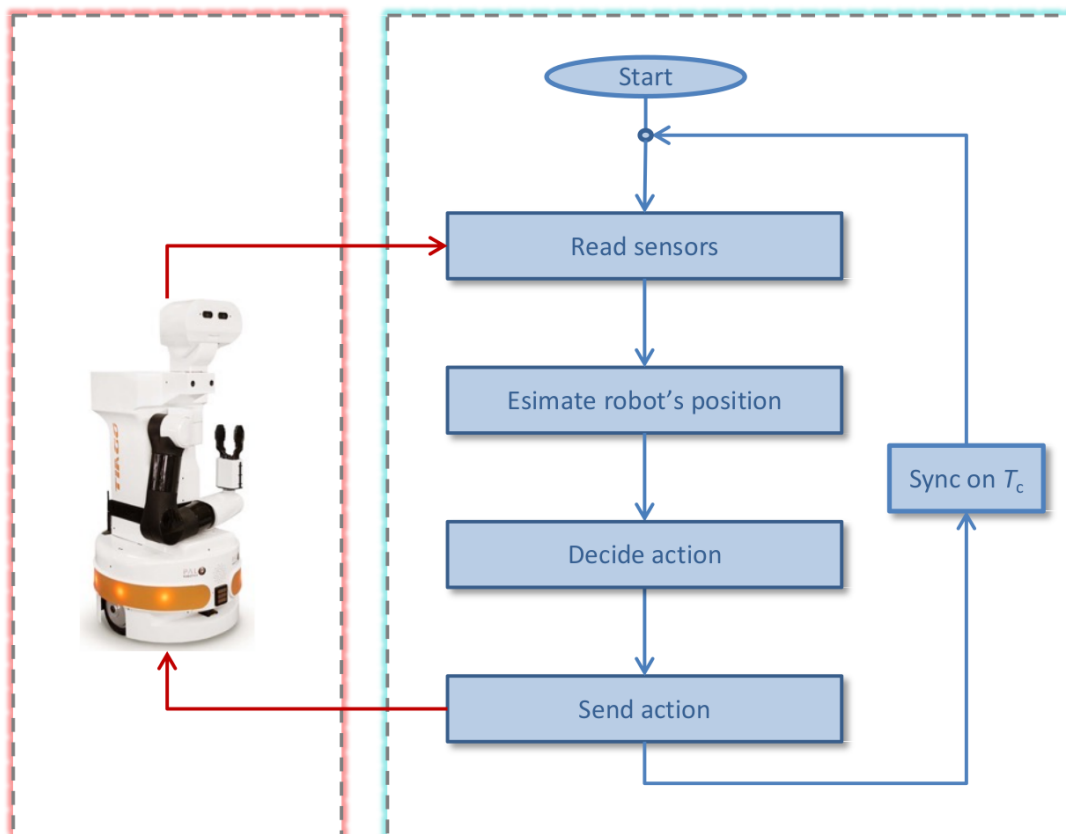


Figure 1: Robot Control Program

1.2 Objects and Data Structures

The chosen design for the RCP is a structure of objects representing our different components:

- RobotGateway implements the "Read Sensors" and "Send Action" boxes.
- Observer implements the "Estimate State" box.
- Controller implements the "Decide Action" component.

Each of this components is defined in a different python file while the `toplevel` implements the general navigation loop which will be in charge of running each of the components during the execution of the simulation.

Notice there are also some useful functions defined for creation of a plot of the trajectory followed by the robot.

1.3 Control and Information flow

As stated on the previous section, the different components of the RCP are implemented using different objects while the overall control is managed by a `toplevel` program. This control and information flow can be seen on figure [2].

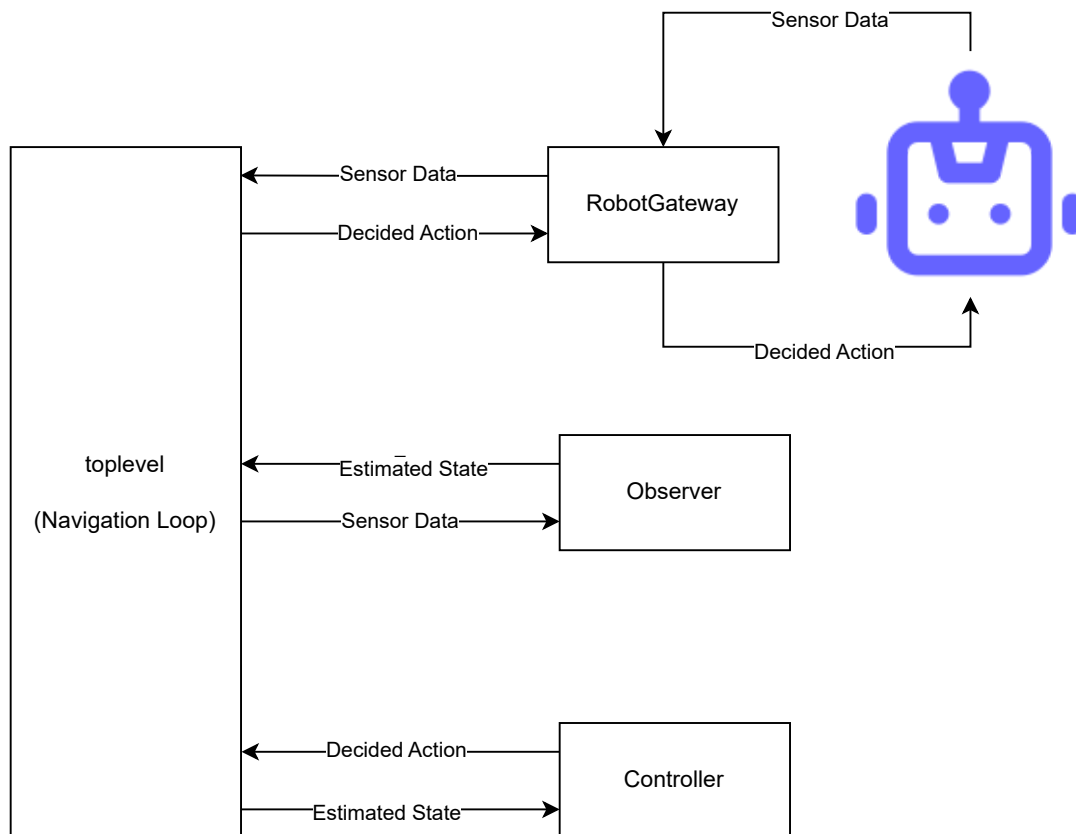


Figure 2: Flow of information diagram

Notice to communicate with the robot we subscribe or publish to the topics provided by our TIAGo robot using the `toplevel` program, which is declared as a ROS node.

1.4 Parameters

Through the whole program we can see the different parameters that had to be read, estimated or defined. We can divided them in the following:

Constant Parameters these parameters are given with the assignment and are intrinsic to the robot. For this assignment we have the wheel radius (r) and the wheel axis (D).

Read Parameters these parameters are read from the robot sensors at each step of the simulation. We read the encoders of the left and right wheels (w_L, w_R), which gives us their position in radians.

Estimated Parameters these parameters are computed using the other two type of parameters and are needed to estimate the position of the robot and therefore to take a decision about the next action. The estimated parameters are the coordinates x and y , as well as the angle of rotation (θ).

For the estimated parameters we apply the general position update given a starting point (the estimated point at previous step given by x_0, y_0 and θ_0).

$$\begin{aligned}x &= x_0 + d_x \cos(\theta_0) - d_y \sin(\theta_0) \\y &= y_0 + d_x \sin(\theta_0) + d_y \cos(\theta_0) \\ \theta &= \theta_0 + \Delta\end{aligned}$$

To compute the distance traveled on each axis we need first to compute the euclidean distance traveled, which can be estimated using the encoders increment d_R and d_L .

$$\begin{aligned}d_R &= r(w_{R0} - w_R) \\d_L &= r(w_{L0} - w_L)\end{aligned}$$

Then, traveled euclidean distance and rotation angle can be computed as follows:

$$\begin{aligned}d &= \frac{d_R + d_L}{2} \\ \Delta &= \frac{d_R - d_L}{D}\end{aligned}$$

Once d is computed we can estimate the distance traveled on each axis:

$$\begin{aligned}d_x &= d \cos \frac{\Delta}{2} \\d_y &= d \sin \frac{\Delta}{2}\end{aligned}$$

1.5 Results Obtained

After executing the simulation setting a certain goal we can see the prints on the terminal for the different increments on each dimension.

```
pose = (3.00, 0.00, 0.00)
pose = (3.00, -0.00, -0.18)
pose = (3.00, -0.00, -0.18)
pose = (3.00, -0.00, -0.18)
pose = (3.01, -0.00, -0.20)
pose = (3.02, -0.00, -0.20)
pose = (3.04, -0.00, -0.20)
pose = (3.05, -0.00, -0.20)
pose = (3.07, -0.00, -0.20)
pose = (3.08, -0.00, -0.20)
pose = (3.10, -0.00, -0.20)
pose = (3.11, -0.00, -0.20)
pose = (3.13, -0.00, -0.20)
pose = (3.14, -0.00, -0.20)
pose = (3.16, -0.00, -0.20)
pose = (3.17, -0.00, -0.20)
pose = (3.19, -0.00, -0.20)
pose = (3.20, -0.00, -0.20)
pose = (3.21, -0.00, -0.20)
pose = (3.23, -0.00, -0.20)
pose = (3.24, -0.00, -0.20)
```

Figure 3: Increment on x axis

```
pose = (3.96, 1.06, 89.96)
pose = (3.96, 1.07, 89.96)
pose = (3.96, 1.09, 89.96)
pose = (3.96, 1.10, 89.96)
pose = (3.96, 1.11, 89.96)
pose = (3.96, 1.13, 89.96)
pose = (3.96, 1.15, 89.96)
pose = (3.96, 1.16, 89.96)
pose = (3.96, 1.17, 89.96)
pose = (3.96, 1.19, 89.96)
pose = (3.96, 1.21, 89.96)
pose = (3.96, 1.22, 89.96)
```

Figure 4: Increment on y axis

```
pose = (3.96, -0.00, 30.44)
pose = (3.96, -0.00, 31.21)
pose = (3.96, -0.00, 32.10)
pose = (3.96, -0.00, 32.87)
pose = (3.96, -0.00, 33.82)
pose = (3.96, -0.00, 34.60)
pose = (3.96, -0.00, 35.49)
pose = (3.96, -0.00, 36.32)
pose = (3.96, -0.00, 37.09)
pose = (3.96, -0.00, 37.98)
pose = (3.96, -0.00, 38.81)
pose = (3.96, -0.00, 39.59)
pose = (3.96, -0.00, 40.48)
pose = (3.96, -0.00, 41.25)
pose = (3.96, -0.00, 42.20)
pose = (3.96, -0.00, 42.97)
pose = (3.96, -0.00, 43.86)
```

Figure 5: Increment on angle

1.6 Discussion

While the robot on the simulation travels x axis the estimated position for that dimension gets incremented (figure [3]). This also happens for y axis and θ while traveling vertically and rotating (figures [4] and [5]). Hence, we could say that as a first impression our estimation is working.

2 Test the quality of your position estimation more extensively

2.1 The problem

For this second task we are asked to perform a more extensive test of our estimation, by plotting the trajectory and applying some well know methods for its correction like the "square trip". We will also play with the given parameters to see how our trajectory estimation changes differ from the ground truth depending on the error.

2.2 Objects and Data Structures

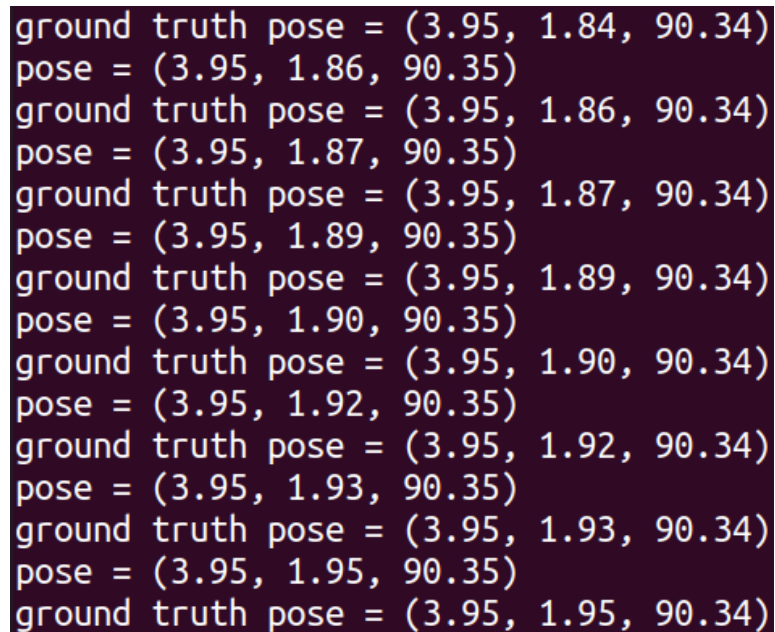
In addition to all the objects and data structures described in the previous task, a pair of arrays were defined to keep track of the history of estimated and ground truth po-

sitions in order to reconstruct the robot's trajectory at the end of the execution. In this way we are able to plot the both trajectories and compare them.

2.3 Results Obtained

We can see the comparison between the estimated position and the ground truth taken from the `/ground_truth_odom` topic on figure [6]

For this test we used the goal (4.0,2.0) which trajectory can be checked on figure [7].



```
ground truth pose = (3.95, 1.84, 90.34)
pose = (3.95, 1.86, 90.35)
ground truth pose = (3.95, 1.86, 90.34)
pose = (3.95, 1.87, 90.35)
ground truth pose = (3.95, 1.87, 90.34)
pose = (3.95, 1.89, 90.35)
ground truth pose = (3.95, 1.89, 90.34)
pose = (3.95, 1.90, 90.35)
ground truth pose = (3.95, 1.90, 90.34)
pose = (3.95, 1.92, 90.35)
ground truth pose = (3.95, 1.92, 90.34)
pose = (3.95, 1.93, 90.35)
ground truth pose = (3.95, 1.93, 90.34)
pose = (3.95, 1.95, 90.35)
ground truth pose = (3.95, 1.95, 90.34)
```

Figure 6: Comparison between estimated position and ground truth position.

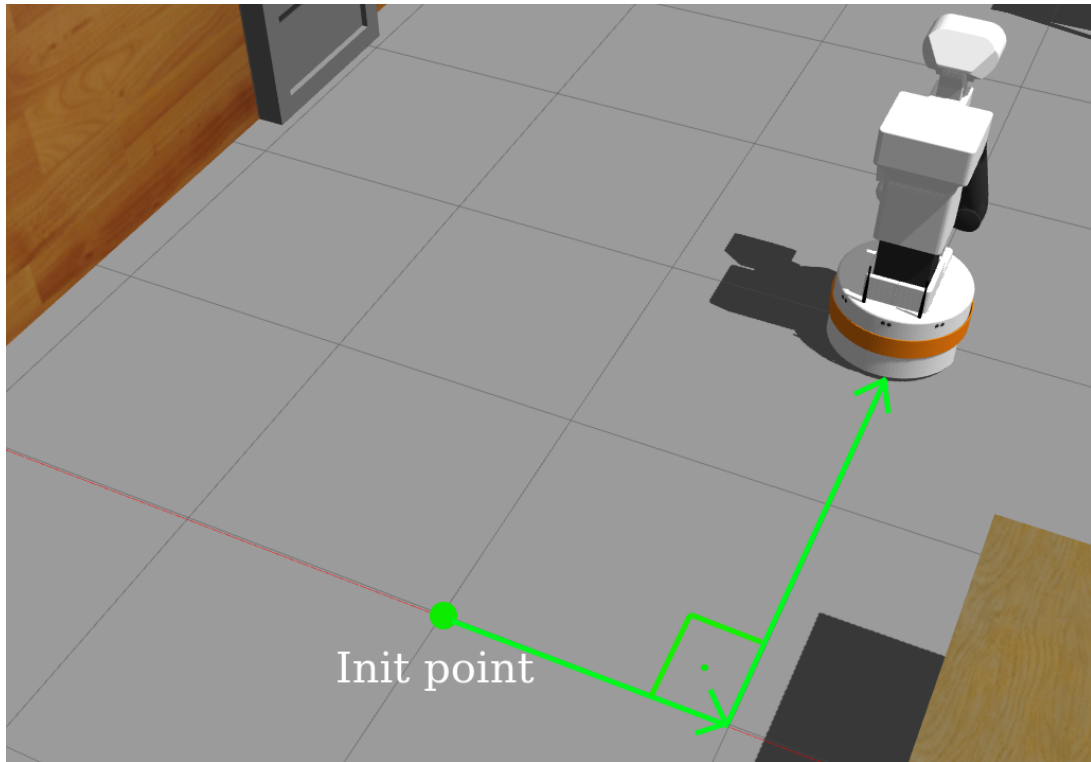


Figure 7: Trajectory followed by the robot for the first test.

Second test results where the "square trip" test was perform can be checked on figure [8] while on figure [9] a close up on the graph can be visualized.

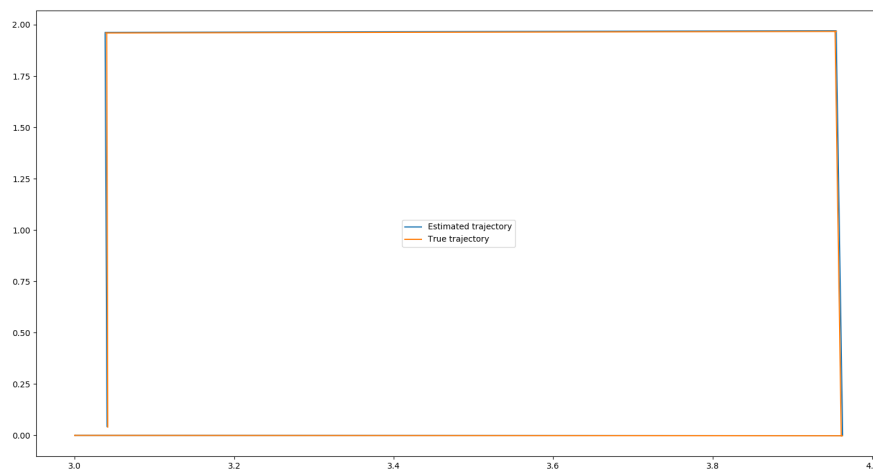


Figure 8: Trajectory comparisons for the "square trip" tests

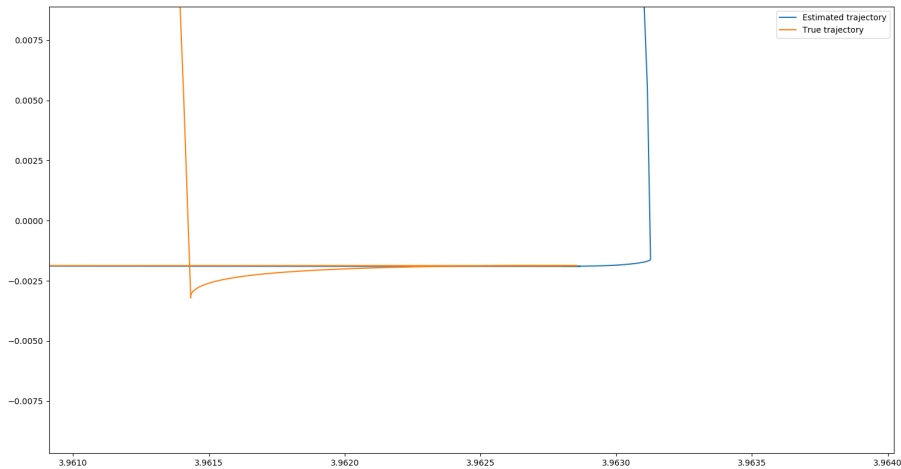


Figure 9: Zoom in to trajectories difference

2.4 Discussion

On our second test, we perform the "square trip" and plot both trajectories to see what is the difference between the initial point and the last point (see figure [8]). Notice the difference between initial and ending position is not very high. Moreover, the difference between estimated and ground truth position at the end of the execution is minimal.

Is interesting to remark that the difference between estimated and ground truth trajectories is only due to the wheel axis and that we can confirm from this that both wheel are perfectly aligned (figure [9]).

3 Implement a linear control strategy to navigate to a given goal position

3.1 The Problem

For our final task of this assignment we are asked to implement a simple linear control strategy. Since until now we have implemented simple dummy strategies where the robot follows an instruction at each step. For this task we implement a way for the robot to compute in its own what action to take depending on where is located the goal position.

3.2 Parameters

As stated on the previous section, we aim to estimate two parameters, v_{lin} and v_{rot} which denotes the linear and rotational velocities. These parameters will be computed following a linear strategy, since we are interested in keeping a constant speed we are mostly aiming to get the rotation sign.

$$\begin{aligned}\Delta_x &= x_t - x_c \\ \Delta_y &= y_t - y_c \\ E_\theta &= \arctan \frac{\Delta_y}{\Delta_x}\end{aligned}$$

In the equation above, E_θ denotes the estimated angle of the target location given by (x_t, y_t) coordinates. (x_c, y_c) represents the current robot coordinates. The angle to the target will be determined by the different with our robot's current angle:

$$\Delta_\theta = E_\theta - \theta_c$$

Then v_{rot} is determined depending on the sign

$$\begin{cases} v_{rot} = 0.1 & \text{iff } \Delta_\theta > 0 \\ v_{rot} = -0.1 & \text{otherwise} \end{cases} \quad (3.1)$$

If the angle difference is close to 0 then the robot will just apply a $v_{lin} = 0.1$ and go straight.

3.3 Results Obtained

The trajectory followed by the robot using the linear control strategy can be seen on figure [10]

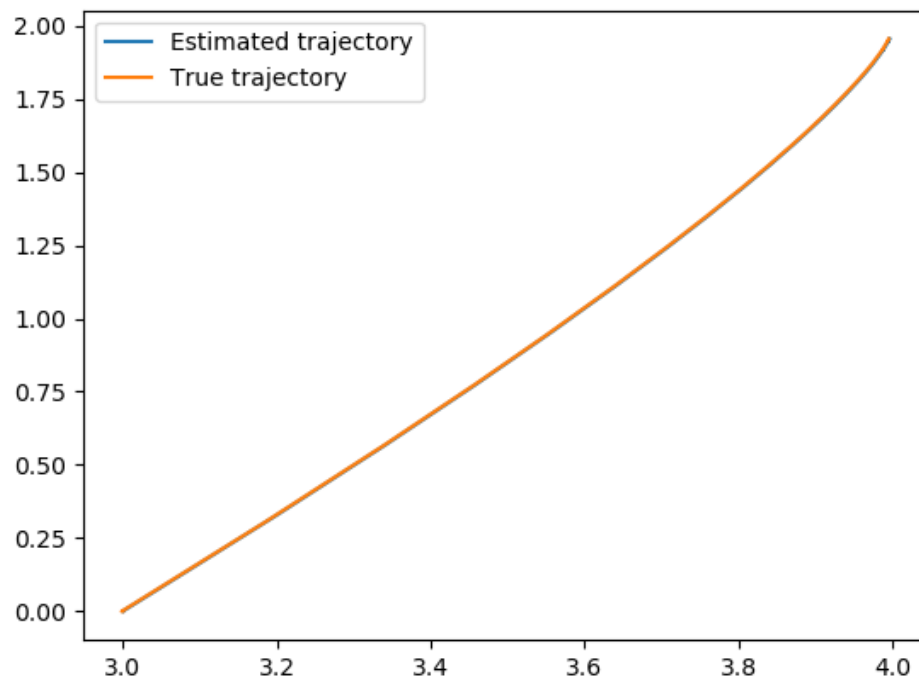


Figure 10: Trajectory followed by the robot using the linear control strategy

3.4 Discussion

Even though this strategy seems to work good, it is obviously not robust to changes in the environment and therefore, it is far away for being intelligent.