Autonomous Mobile Robots
Homework 1

Ahmed Mohsen ALi

September 2022

Task Description

In this task, I try to calculate the trajectory planning of a differential robot using analytical and simulation methods. There are four different scenarios for the robot, each with different initial conditions. The setup is an empty environment, free from any obstacles. The software used is :

- ROS2 humble on Ubuntu 22 (Docker container)
- Python3 scripts
- Gazebo and Rviz for simulation

Problem Solution

The trajectory of each scenario is calculated through two distinct approaches:

1. Analytical solution is provided using python scripts. It is calculated from forward kinematics equations discussed in the lectures.
2. Simulation data are generated from odometry topic published by Gazebo (figure 1).

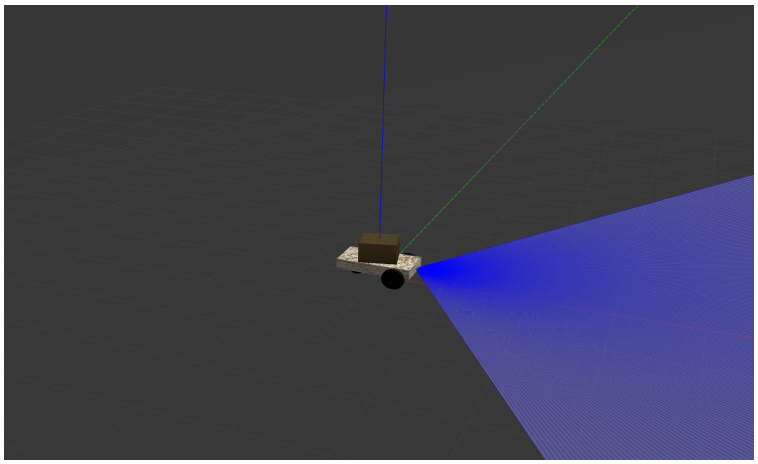


Figure 1: Gazebo Simulation

Also, it is worth mentioning the parameters used for the simulations which are:

- $T_s = 0.033\,s$
- $r = 0.04\,m$
- $L = 0.08\,m$
- Total simulation time was 5 seocnds

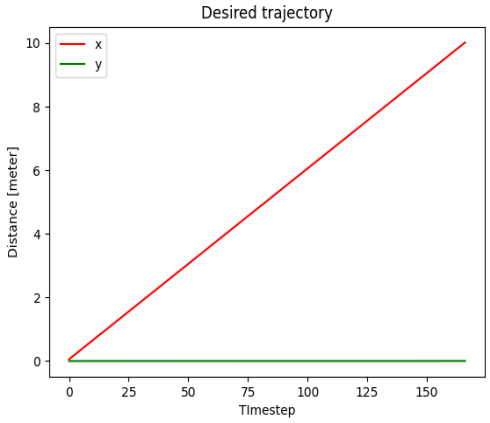
The results for each scenario are presented and discussed in the following subsections.

Scenario 1

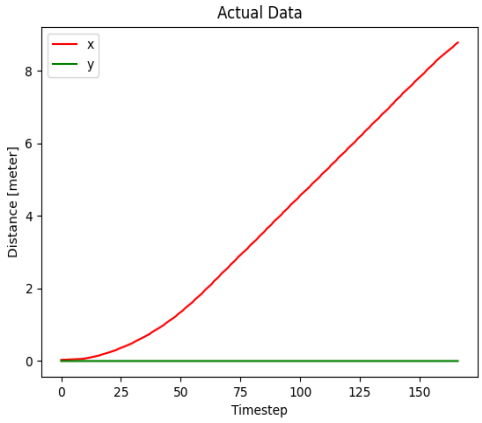
Initial conditions:

- $v(t) = 0.5\,m/s$,
- $\omega(t) = 0\,rad/s$

Results:



(a) Desired trajectory generated from equations



(b) Actual trajectory generated from Gazebo

Figure 2: Trajectory data for scenario 1

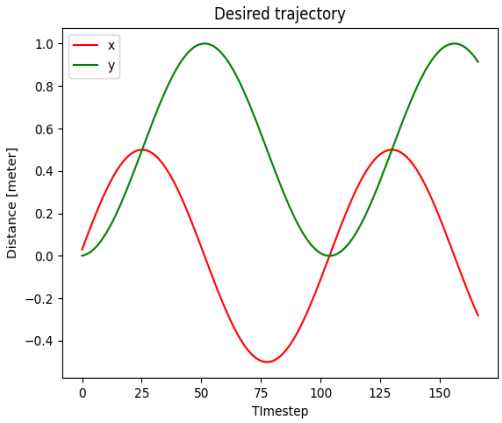
Discussion:
As shown in figure 2, the desired trajectory is a perfectly straight line (figure 2a) since only linear velocity is provided. However, there is noticeable difference in actual trajectory (figure 2b). These variations, especially at the start, are due to other factors not considered in the equations such as static and dynamic friction, air resistance, and slippage.

Scenario 2

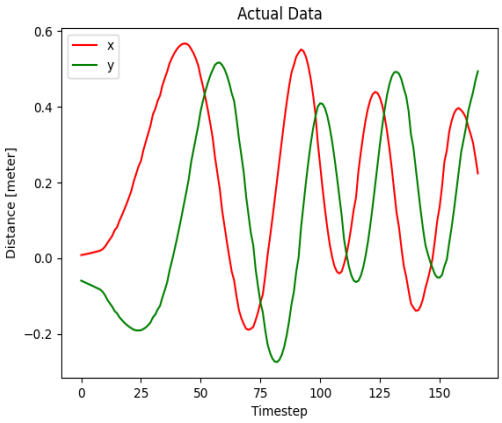
Initial conditions:

- $v(t) = 1\,m/s$,
- $\omega(t) = 2\,rad/s$

Results:

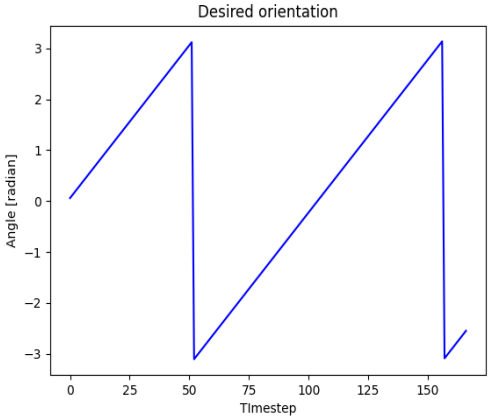


(a) Desired (x,y) positions generated from equations

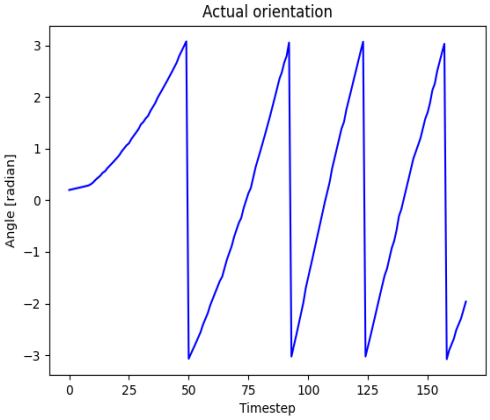


(b) Actual (x,y) positions generated from Gazebo

Figure 3: (x,y) positions data for scenario 2

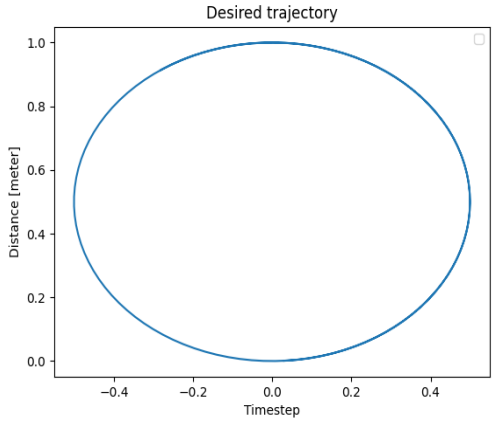


(a) Desired orientation generated from equations

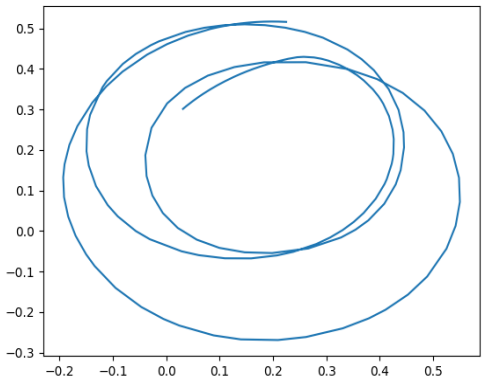


(b) Actual trajectory generated from Gazebo

Figure 4: Orientation data for scenario 2



(a) Desired trajectory generated from equations



(b) Actual trajectory generated from Gazebo

Figure 5: Trajectory data for scenario 2

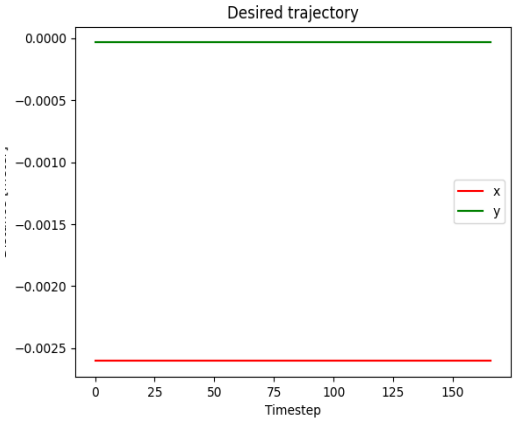
Discussion:
As shown in figure 3, the expected (x,y) positions differs a lot in magnitude and oscillations from actual (x,y) positions. This is mainly because the robot is treated as single point of mass in equations. However, this is not the case in reality where the object in 3d and have a center of mass that can be affected by moments during angular movements. Figure 4 and figure 5 show the variations in orientation and trajectory respectively. Although the desired trajectory was a circle, the actual one was clearly different, with multiple circles of different centers.

Scenario 3

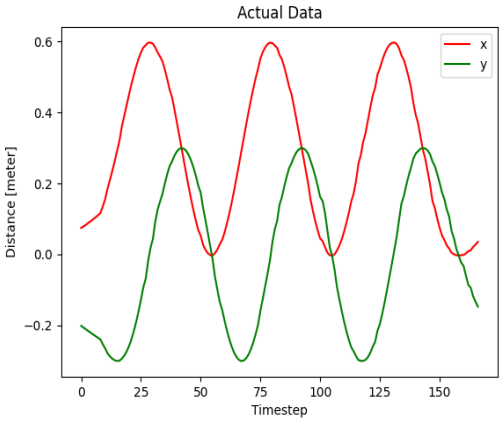
Initial conditions:

- $v(t) = 0\,m/s$,
- $\omega(t) = 2\,rad/s$

Results:

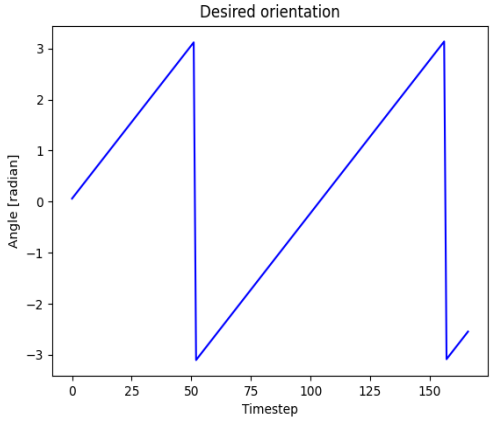


(a) Desired (x,y) positions generated from equations

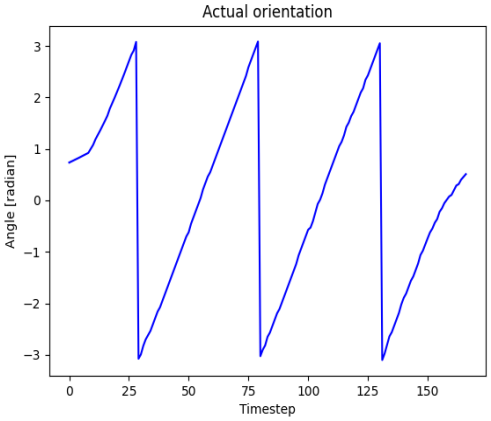


(b) Actual (x,y) positions generated from Gazebo

Figure 6: (x,y) positions data for scenario 3



(a) Desired orientation generated from equations



(b) Actual trajectory generated from Gazebo

Figure 7: Orientation data for scenario 3

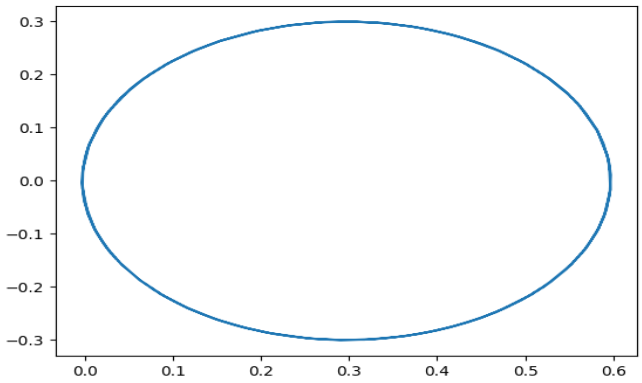


Figure 8: Actual Trajectory data for scenario 3

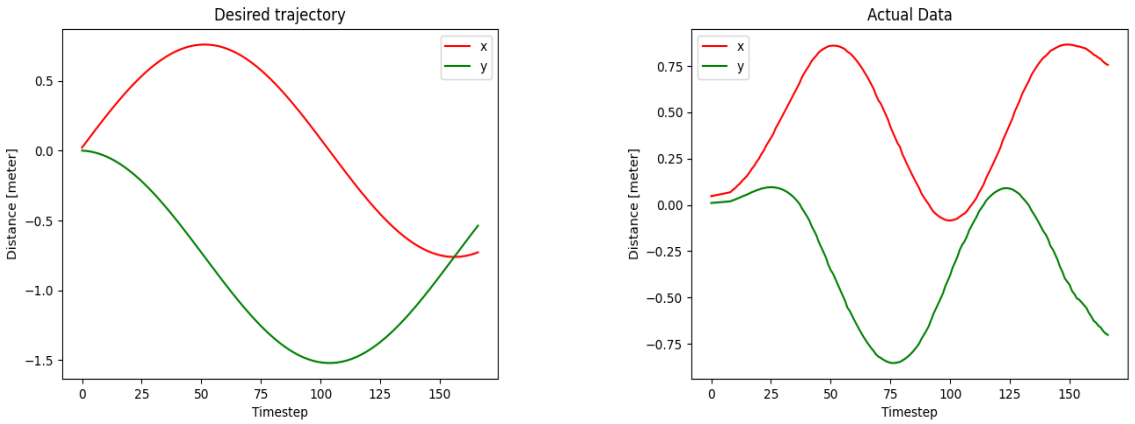
Discussion:
As shown in figure 6, the expected (x,y) positions should remain at zero since no linear velocity is provided but the actual position of robot slightly changes (around 60 cm deviation in each axis). Since the real center of mass may no be the same as the center in instantaneous center, these deviations will be recorded from odometry sensors. Figure 7 and figure 8 show the variations in orientation and trajectory respectively.

Scenario 4

Initial conditions:

- $\omega(t)_L = 20\,m/s$,
- $\omega(t)_R = 18\,rad/s$

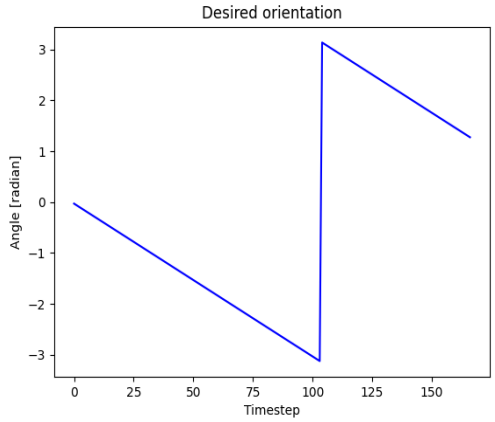
Results:



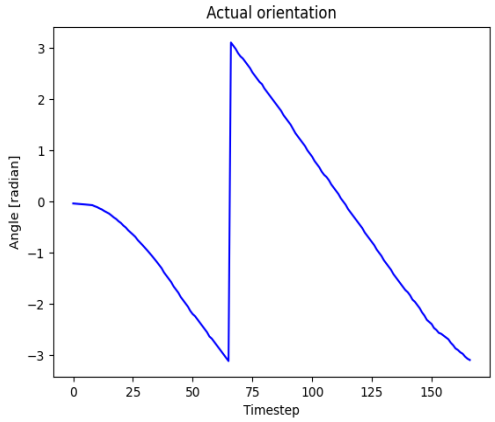
(a) Desired (x,y) positions generated from equations

(b) Actual (x,y) positions generated from Gazebo

Figure 9: (x,y) positions data for scenario 4

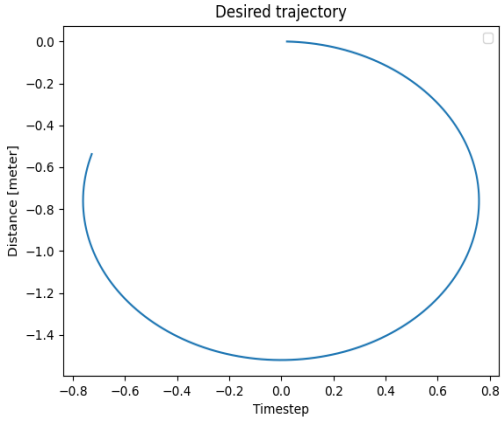


(a) Desired orientation generated from equations

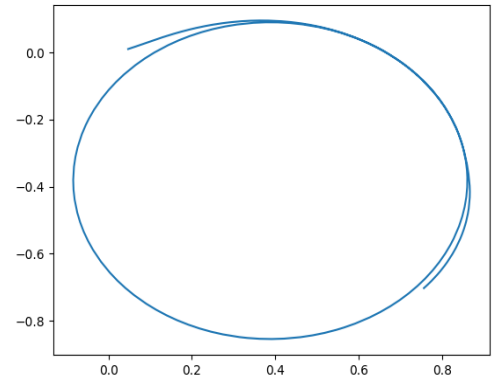


(b) Actual trajectory generated from Gazebo

Figure 10: Orientation data for scenario 4



(a) Desired trajectory generated from equations



(b) Actual trajectory generated from Gazebo

Figure 11: Trajectory data for scenario 4

Discussion:
In this scenario, the inputs are changed. Instead of providing linear and angular velocities of the robot directly, only angular velocity of left and right wheel are provided. So the kinematics equations are extended to convert these inputs into $v(t)$ and $\omega(t)$.
As shown in figure 9, the expected (x,y) positions are different from actual ones. Figure 10 and figure 11 show also the variations in orientation and trajectory respectively.

Important remarks

- The error between desired and actual robot pose occurs since there are some factors neglected in kinematic model of differential drive. For example, the effects of friction and robot mass are not considered.

- To eliminate the error between desired and actual data, a feedback controller (such as P controller) has to be provided.
- In cases of high speed, dynamic model should be used instead of kinematics.

Technical Note:

- There was a problem on how to get the desired and actual data from one node at the same time since desired data are calculated from the script and actual data are received from topic /hagen/odom. To overcome this problem, I created two python scripts and ran them simultaneously to get these data. This solution is in the attached source code files.
- **To run the code:** After installing this git repo, add the attached two source codes to the contents of `your_workspace_dir/src/autonomous_mobile_robots/hagen_control/hagen_control`. After that run the following commands in separate terminals [in the same directory mentioned previously]:
  - `ros2 launch hagen_gazebo hagen.launch.py world:=hagen_empty.world`
  - `python3 hagen_control_strategy.py`
  - `python3 sim_node.py`