CSE 276A - Introduction to Robotics

HW1

Drishti Megalmani A59001657
Sai Ashritha Kandiraju A59001630

Video Link: https://www.youtube.com/watch?v=tGXIXWXOTAY

CALIBRATION

We determined the desired linear speed of the robot by running multiple experiments and recording the distance it covered in each run. We experimented with varying speeds to motorRun and finalized on a vehicle linear velocity of 0.25 m/s. Similarly, we calculated the desired angular velocity of the vehicle by modeling its angular displacement in a given time and decided on a value of 1.20 rad/s.
We started off with the same initial speed in motorRun for all the 4 wheels. Our experiments showed us that the number of rotations (clockwise and counterclockwise) for the same speed differed slightly for all the 4 wheels. To compensate for this difference, we manually calibrated the motors by trial and error.

DESIGN OF CONTROLLER

To move the robot to the desired target pose, our control algorithm first orients the robot in that direction, moves to the destination using a forward motion and then rotates to the desired orientation using clockwise/counterclockwise rotations. Thus a design decision we took was to always perform only forward motion and rotations instead of backward and sliding.

Let's consider current waypoint w (x, y, θ) and target point w* (x*, y*, θ^*). To perform the movement from w to w*, we divided the movement for each waypoint into 3 separate functions:

1. Calculate the initial rotation angle required to head in the direction of w* waypoint from current waypoint w. This was done using the following formula:

   ```
   theta_heading = math.atan2(y* - y, x*- x) - θ
   ```

   We also update a global current angular orientation of the robot as required. Based on the sign of `theta_heading`, we make a decision to rotate the robot clockwise or counterclockwise.
2. After the robot rotates by that angle pointing in the direction of w*, we find the euclidean distance between w and w* and perform a linear forward translation to the waypoint. We find the euclidean distance using the formula:

   ```
   dist = math.sqrt((y* - y) ** 2 + (x* - x) ** 2)
   ```

3. We then perform the final rotation operation, by calculating the difference between the current orientation of the robot and θ^* (the final orientation the robot needs to be in) i.e:,

   ```
   rel_angle = θ* - current_theta
   ```

   Noting here that the current_theta here is not just θ but is an updated value from step 1 (inclusive of theta_heading). Based on the sign of `rel_angle`, we make a decision to rotate clockwise or counterclockwise.

Each of these operations are made to run for a fixed duration that is calculated based on the desired angle/distance to be moved and the calibrated angular speed and linear speed. This is calculated using:

```
duration_angular = abs(angle) / angular_speed
duration_linear = distance / linear_speed
```

By performing these three operations in their respective order, we manage to perform the navigation of the robot through all the waypoints provided.

IMPLEMENTATION OF CONTROLLER

To reach every new waypoint, we perform three operations (a rotation, a forward translation and another rotation). We implemented this by doing the following.

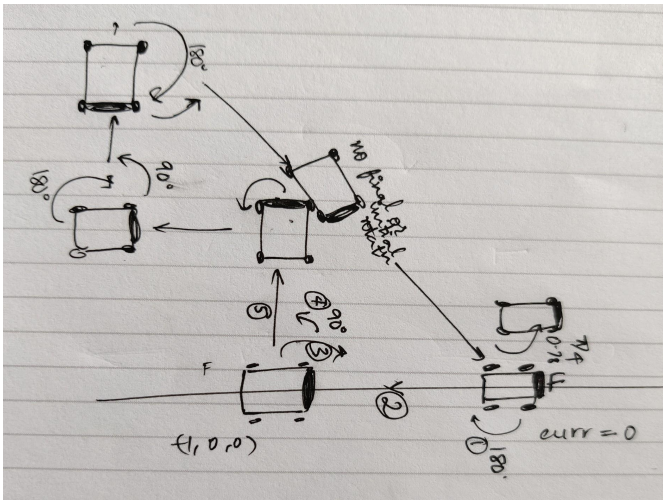
We modified `key_joy_node.py` to change the source of input from the keyboard keys to the txt file. We added 3 new functions to calculate the initial rotation angle, the translation distance and the final rotation angle. We modified the `run()` function to perform these three operations for each waypoint in the csv file. After the duration for each operation is calculated (as explained in the Design section), we send a publish message for the key from the `key_to_joy()` function:

```
self.pub_joy.publish(joy_msg)
time.sleep(duration)
```

We use a `time.sleep()` to make sure to run the desired operation for our calculated duration. We are hence, using the given mpi_controller with the implemented Kinematic model. We set the following speeds for the straight motion and rotation in mpi_control_node. These speeds ensured a minimal drift.

```
v_max_default_straight = 50
v_max_default_rotate = 50
```

The coordinate system convention that we used is given in the picture below. The robot starts at 0,0,0 and the front of the robot points towards the positive x-axis.



OBSERVATIONS

The robot correctly navigates to all the waypoints covering the desired distance. It also orients itself as required after reaching a target waypoint. However, even after calibration we've observed that the robot slightly deviates from its path after a clockwise rotation. This produces a slight drift which cascades throughout its motion resulting in the robot not reaching the exact same position that it started off at. We observed that this difference is not huge and is off by about 20 cm from the origin on the y-axis).

Since we employed an open-loop design for the control algorithm, we are unable to correct the impact of the drift in the system. In the future, when we modify this to a closed-loop system, we could use the feedback control from sensors and avoid this difference.

CHALLENGES

1. Since we both had a Mac M1 we had difficulty flashing Rb5 with the help of a docker. We spent about 2 days getting this resolved and finally we were able to flash the system with a TA's help.
2. For one of our robots we received a faulty connector that had to be replaced.
3. Since the right orientation of the wheels wasn't clearly mentioned in the assembly instructions, we spent some time figuring out why the motors didn't move as desired. We fixed this after rearranging the wheels as required.
4. The motors weren't moving for lower speed values.
5. We observed that the MegaPi motorRun function did not work deterministically. Sometimes for the same speed the motors did not rotate.
6. We spent a lot of time trying to figure out a control algorithm where we used both the forward and backward motions for linear movements. We ended up with a lot of cases where it was ambiguous to determine the direction of motion.

CONTRIBUTION

Both of us contributed equally on all sections of the homework (from calibration, design thinking, implementation, testing and report).