CSE 276A - Introduction to Robotics

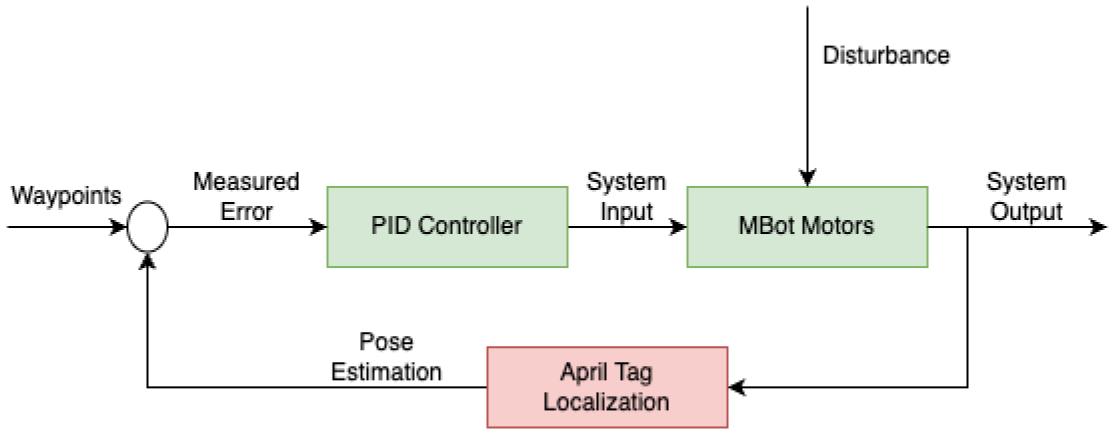
HW2

Drishti Megalmani A59001657
Sai Ashritha Kandiraju A59001630

Video Link: https://youtu.be/HjH1G39pJKs

In this homework, we aim to convert the open-looped controller we built in homework 1 into a closed loop system by using the camera module to improve the localization performance. The visual feedback comes from a known landmark position (april tag) which enables us to estimate the robot's relative pose with respect to the landmark thus facilitating accurate movement of the robot.

METHODOLOGY



To implement the closed loop system, we perform the following:

1. **Localization**
   We added a new localization node (localization_node.py) which subscribes to the */apriltag_detection_array* and publishes the current pose estimate of the robot in world coordinates to */current_pose*. To implement this node, we first calibrated the camera in robot frame transformation matrix (*rTc*) and the pose matrices (*pose_tag*) for the april tags we used. We used 2 april tags for visual feedback. For every msg.detections in the */apriltag_detection_array,* we found the current pose by doing the following transformations:

   Given: rTc is our defined transformation for camera in robot frame.
   Pose_tag is our defined matrix for all the locations of the april tags
   cTa is the april tag transformation matrix in camera frame obtained from april_detection_node

April tag in robot coordinates:

rTa = rTc @ cTa

Robot in april tag coordinates:

aTr = np.linalg.inv(rTa)

April tag in world coordinates:

wTa = pose_tag[apriltag_id]

Final Robot in world coordinates:

wTr = wTa @ aTr

@ - matrix multiplication

2. **Controller**
   In this file, we replicated the hw1_solution.py (PID Controller) file published as the solution for homework one and made some modifications to incorporate the visual feedback. We subscribed to */current_pose* which gave us the current pose of the robot in the world coordinates. We updated the waypoints to the new waypoints given in the question. For each waypoint, we publish a twist message to the /twist node. We added an additional update statement within the error check loop between the current state and current way point. We check if there is visual feedback from the camera via */current_pose* and find the corresponding current state in (x, y, θ) format. We wrote our own *matrix_to_euler*() function that converts the 3x3 rotation matrix in the current_pose 4x4 matrix into the 3 euler angles in the 3 axes. We use the yaw angle for our required θ. The x and y come from the translation matrix from the current_pose. We then update our current position of our robot to (x, y, θ).

   If there is no landmark in view of the camera i.e, if msg.pose (from the localization node) doesn't return anything, we fallback to the usual update operation (open-loop) of the PID controller.

CALIBRATION

Camera Calibration

We first calibrated the camera using a checkerboard of size 8x6 and square size of 2.5cm or 0.025m. We used the larger camera_0 for our whole feedback system with more than 100 samples. The obtained camera matrix after calibration is as follows:

```
635.468668 0.000000   958.618210
0.000000   632.846573 533.960967
0.000000   0.000000   1.000000
```

The distortion obtained for this camera matrix was as follows:

```
-0.039503 -0.000938 0.001796 -0.001603 0.000000
```

April Tag Placement

We decided to use 2 april tags. The world coordinates of these tags were at:
April Tag id = 8: 1m away from the first waypoint approximately (1,0,0) in the z axis wrt April tag. Its world coordinate location was at (2,0,0). The transformation matrix for this tag is:

```
 0   0   1  2.05
-1   0   0  0.015
 0  -1   0  0.15
 0   0   0   1
```

April Tag id = : 5 placed at world coordinates approximately about (0,2,0). This was 1m away from our waypoint (1,2,pi) in the negative z-axis wrt April tag. The transformation matrix is:

```
 0   0   1  -0.2
-1   0   0  2.3
 0  -1   0  0.15
 0   0   0   1
```

We manually estimated the positioning of the April tags and ran the localization node to get the corresponding world coordinates of the robot and we made adjustments to the positioning until we achieved the desired position of the robot and marked that as our waypoint.
We chose these two points and these orientations to facilitate the motion to the two points. For its motion from (0,0,0) to (1,0,0), the first April tag was in the frame. For half the motion from (1,0,0) to (1,2,pi), the first way point was in the frame of the camera output and since the robot rotates to achieve the orientation of pi, for the second half of the path, the 2nd april tag was in the frame. For the final stretch back to (0,0,0), it gets feedback from both april tags again. Since, almost at all times either one of the april tags is in the camera frame for visual feedback, we didn't feel the need to any more.

PID Tuning

We started our experimentation with PID values 0.02, 0.005, 0.005 given in the Hw1 controller. We observed that for low values of P (< 0.01) the motor wheels did not move (due to the friction on the carpet). When we increased P to values greater than 0.02, the velocity was high which resulted in the bot overshooting its target and spending a lot of time trying to correct the offset. We then manually tested for values between 0.01 and 0.02 and finalized 0.0185 as it provided the best performance.

For higher values of I (greater than 0.002), we observed that it resulted in non-smooth motion of the robot (jerky motion). For extremely small values of I in the range 0.001 and lower, the motor did not get enough power to rotate as required. We then manually tested and finalized on 0.0015.

For values of 0.1 and higher for D, we observed that the motion of the robot was not smooth. For values below 0.05, the motion was very smooth and the robot spent time correcting the errors that resulted due to off-shooting its target. We experimented with values between 0.05 and 0.1 and observed best performance for the value of 0.09. The final PID values are provided in the table below

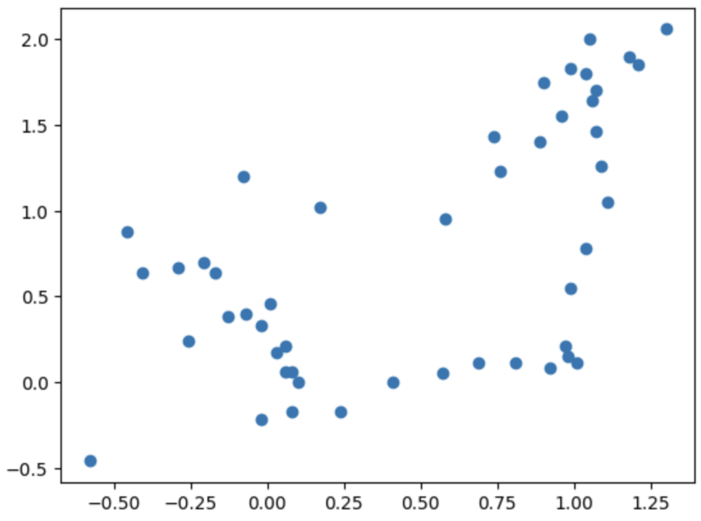
P	**I**	**D**
0.0185	0.0015	0.09

OBSERVATIONS

1. For a very low error, the robot oscillated quite a bit and struggled to achieve such an accuracy. This could be due to the small errors in calibrating the positions of the april tags and waypoints.
   We tried values from 0.05 to 0.2. For 0.1, we found that trade off between the accuracy and the drift/ oscillations are optimal.
2. Translation and Rotation Errors at each way point:

Waypoint	Translation Error	Rotation Error
(1,0,0)	0.0986	0.0171
(1,2,pi)	0.0979	-0.21946
(0,0,0)	0.0938	0.0362

3. Plot of movement. We plotted every 10th point (timestep = 0.05). We see that it reaches all the points accurately.



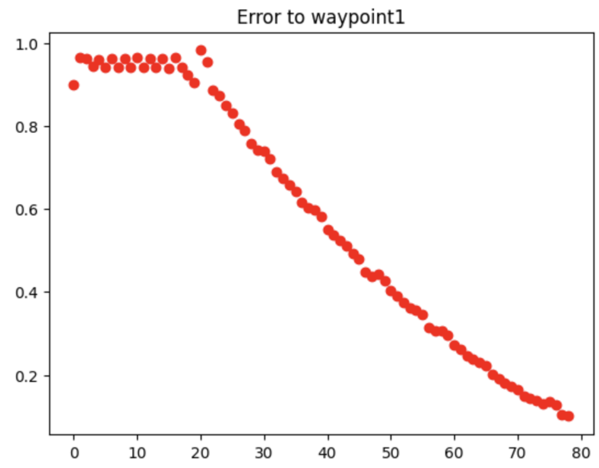
4. The total distance covered by the robot is **12.7446724764.** We calculated this using the below formula where Δx_i represents the x-distance between two consecutive points in the bots motion and Δy_i represents the y-distance between two consecutive points in the bots motion.

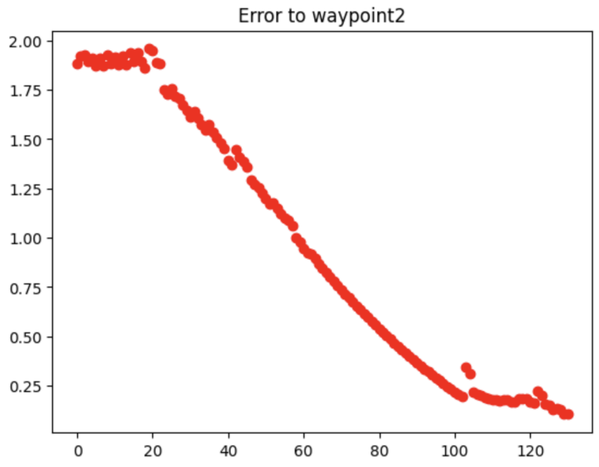
$$\sum_i \sqrt{\Delta x_i^2 + \Delta y_i^2}$$

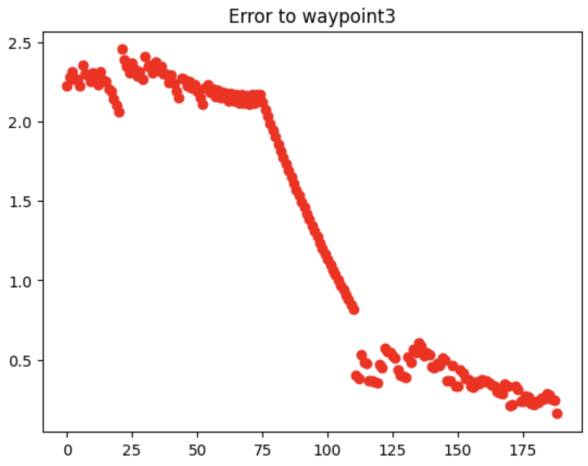
5. The total orientation made by the robot is **16.4616466376.** We calculated this using the below formula where $\Delta\theta_i$ represents the difference in θ between two consecutive points in the bots motion

$$\sum_i |\Delta\theta_i|$$

6. Error wrt each waypoint for smoothness. We see that for the first two points, the error is decreasing almost linearly showing that the motion was smooth. But for waypoint 3, the motion is not as smooth. This is due to the robot not having visual feedback for a short duration which explains the deviation in the path.







Overall, we believe that our algorithm is providing good results and the robot is able to reach each waypoint accurately compared to the solution in homework 1.

CHALLENGES

1. Since both of us had M1 Macs, we weren't able to set up rviz successfully. Due to this, we took a lot of time visualizing the coordinate systems of the world and camera and understanding the transformations required.
2. We faced difficulties with calibrating the P,I,D values for the PID Controller for compatibility for a carpeted floor.

CONTRIBUTION

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