Homework-2

CSE 276A Fall'24 — Introduction to Robotics

Pranav Malpure — **PID.: A69030926**

Akshar Tumu — **PID.: A69032632**

Closed-loop Controller

Our closed-loop controller consists of the following main components and flow of information:

- **Navigation Algorithm**: This component takes the **current state** of the robot and the **target state** as the input, calculating the error (difference) between them. Then, it uses a calibrated PID (Proportional-integral-derivative) controller to figure out the **target velocity** (linear and angular) that the robot has to move with for the next time step.
- **Kinematic controller**: This consists of a kinematic model that converts the input target velocity to the **control signals** that need to be sent to each wheel motor of the robot.
- **Robot (Actuator)**: We use a Qualcomm RB5 robot which is a 4-wheel mecanum robot. It takes in the **control signals** from the kinematic controller and passes them to the corresponding motors of the wheels for the motion to occur.
- **Sensor (Camera)**: The sensor consists of the camera on the robot which looks for an April tag in its view and detects its position and orientation.
- **Estimate Current State logical program**: This takes the **April tag detection** of the sensor as the input. Then, we run a logical program that estimates the current state and orientation of the robot given the April tag detections. If no April tag is detected, the logical program uses the **update value** calculated by the PID controller for estimating the current state.

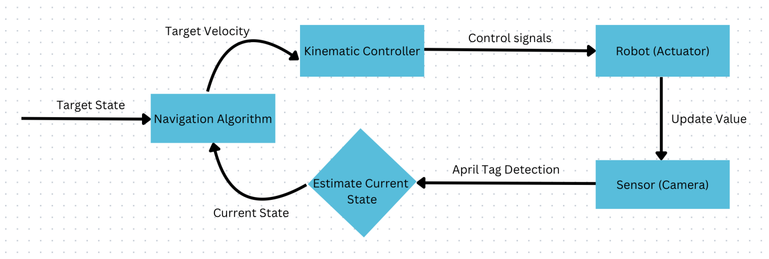

Figure 1: Flowchart of the closed-loop controller.

Landmarks/April Tags Description

We work with 5 landmarks placed at the following positions: (the format being (x, z, theta) in the world frame)

1. (1, 0, 0) - helps the robot traverse to the (0.5, 0, 0) waypoint linearly.
2. (1, 1, 0) - Helps the robot move to the (0.5, 1, π) position when the robot loses sight of landmark 1. It also helps in locating the robot when it turns π radians after reaching the (0.5, 1, 0) position
3. (0.5, 1.5, $\pi/2$) - Helps in locating the robot when it turns π radians after reaching the (1/2, 1, 0) position and it loses sight of landmark 2 while turning.
4. (-0.41, 1, -π) - Helps the robot in completing its π radian rotation at (0.5, 1, 0) when it loses sight of landmark 3. It also helps the robot traverse back to (0, 0, 0)
5. (-0.41, 0, -π) - Helps the robot navigate back to (0, 0) coordinates and then helps the robot rotate back to 0 radians orientation.

There is a 6th april tag seen in the video, but it is supposedly redundant for the fact that the robot never turns towards it.

NOTE: While the robot rotates back to (0, 0, 0) orientation, even landmarks 1-4 come into the robot's view intermittently, ensuring that the robot completes its rotation without losing sight of landmarks/april tags at any stage.

Camera Calibration

The camera mounted on the robot has a certain level of inherent distortion. Hence, calibrating the camera to handle the distortion while projecting the model into 2D homogeneous coordinates becomes crucial. For this, we made use of the OpenCV's 'cameraCalibrate' function.

For this, we first obtained a chessboard of 8x8 dimension and having sides of length 6.4 inches. Then, we captured 18 images of the chessboard in various orientations using the robot's camera. Then, we made use of OpenCV's 'cameraCalibrate' function to obtain the intrinsic matrix and the distortion coefficients of the camera. Feeding the obtained values and coordinates to the 'April detection' package helped correct the distortion in the camera.

Robot Pose Estimation

Assumptions

We assume the location of the camera to be the location of the robot. This is because the camera is mounted on the robot itself. The camera is very close to the center of the robot and the z-axis of the camera is perfectly aligned with the front of the robot.

We assume that all the movements of the robot happen in the X-Z plane of the world frame. From the (0, 0, 0) position in this plane, the +X axis is forward and the +Z axis is to the left.

April Tag error correction

We observed that the April tag was accurate but had some kind of error when we compared that to the original position of the camera. To map these errors, we made observations at the following 5 points:

Actual Camera Pos:	95	62	49	26
April Tag detected pose:	100	75	50	25

Table 1: April Tag-z observations(in cm)

We can see that the above points can be closely represented linearly by the following eq:

$$Z_{actual} = Z_{apriltag} - \frac{Z_{apriltag} - 0.375}{0.125}$$

We use this to correct for the error observed in April tag detection.

Coordinate transformation

When the sensor (camera) detects an April tag, it receives the following information - the position of the origin (center) of the April tag (x, y, z) and the spatial orientation of the April tag (x, y, z, w), both w.r.t the coordinate frame of the camera. Our job is to now convert these coordinates into the world frame.

Now, we calculate the pitch of the April tag using the x,y,z,w from its orientation as follows:

$$\text{pitch} = \tan^{-1}\left(\frac{2(wy - xz)}{1 - 2(y^2 + z^2)}\right)$$

The pitch represents the angle that the axis of the camera makes with the axis of the April tag. Since we know the orientation (angle) of the April tag in the ground frame, we can find the orientation of the robot in the ground frame as follows:

$$\theta_{\text{RO}} = \theta_{\text{ATO}} + \theta_{\text{ATR}}$$

Where θ_{RO} is the orientation of the robot in the world frame, θ_{ATO} is the orientation of the April tag in the world frame and θ_{RAT} is the angle between the robot and April tag's axes

Once we find θ_{RAT}, we can use it to find the location of the April tag w.r.t to the robot in the world frame (x, z) [(x', z') is the location of the April tag w.r.t the robot in the robot frame.] by coordinate axes rotation as follows:

$$x = x' \cos\theta_{RO} - z' \sin\theta_{RO} \qquad z = x' \sin\theta_{RO} + z' \cos\theta_{RO}$$

From this, we get the location of the robot in the world frame as follows:

$$R_{\text{RO}} = R_{\text{ATO}} - R_{\text{ATR}}$$

Where R_{RO} are robot coordinates in the world frame, R_{ATO} are April tag coordinates in the world frame, and R_{ATR} gives the position of the April Tag w.r.t the Robot in the world frame of reference.

Handling cases when no April tag is detected

Whenever our algorithm doesn't detect any April Tag due to no tag being present in the camera's view, it uses the update value from the PID controller to update the current state of the robot.

Results

Car Motion and Smoothness

We designed the car to move in a stop-and-start fashion with a time gap of 0.5 seconds between each movement. We separate the translation and rotational movements into two separate components. Hence, the car first translates to the target waypoint, following which it rotates to match the required orientation. The translation error that may be caused during rotation is again corrected by the algorithm before moving on to the next waypoint.

The car moves in a rather smooth trajectory between two waypoints, with minor corrections to make sure that the robot stays on its course.

Distance Metrics

Due to the limited availability of space, we scale down the coordinate values by 50% for our experiments.

The total distance traveled by the robot is: 4.828 m

The rotation achieved by the robot in the path is: 8.36 radians

Since the robot underwent oscillations at multiple points to correct its angle, the total rotation achieved by the robot should be greater than 2π, which is evident here.

Error metrics

The rotation and translation error from each waypoint is as follows:

- Waypoint 1 (0, 0, 0): The translation and rotation error is 0 since this is the initial pose of the robot too.
- Waypoint 2 (0.5, 0, 0): Translation error: 0.02752 m, rotation error: 0.00447 radians
- Waypoint 3 (0.5, 1, π): Translation error: 0.01815 m, rotation error: -0.00518 radians
- Waypoint 4 (1, 0, 0): Translation error: 0.02969 m, rotation error: 0.00673 radians

The average translation error of the car at the three points is 0.02512 m

The average rotational error of the car at the three points is 0.00966 radians

Car Trajectory

The figure below shows the trajectory of the car as estimated by the algorithm. We plot a point every time the car makes a discrete movement.

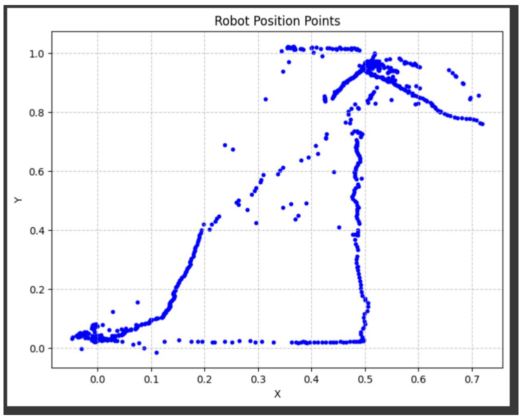

Figure 2: Plot of the trajectory of the robot

Some erroneous points were also recorded as the current state by the algorithm. However, these erroneous states only occurred momentarily and were corrected by the algorithm promptly to make sure that the car never took a big step toward them.

Overall Performance

By observing the graphed trajectory of the car and the average rotational and translation errors of the car at the three waypoints, we can observe that the car sticks to the course set by the waypoints to a large extent, with minor deviations in the path which are corrected for by the algorithm.

Using a similarly calibrated open-loop model on the same surface (carpet), we observed a translation error of 0.5 meters and a rotational error of 0.78 radians from the destination waypoint. By incorporating the April tags and closing the loop, the translation and rotational errors from the destination waypoint have reduced by 0.475 m and 0.77 radians respectively. This shows a significant improvement in the model performance that has been brought upon by closing the loop.

Youtube Unlisted Video Link

Click ***here*** for the link: https://youtu.be/Yfs28kUWw8U?si=0ZHPRdpy6k9iyfrd