CSE 276A HW1

Tingyu Shi(A59023729)
Jiajun Li(A16635772)

October 17, 2024

1 Video URL

URL: https://youtu.be/wFwZRhoc9JA

2 Calibration Strategy

In this part, we investigated the relationship between the motor value and wheel's angular speed for each wheel. We used the following steps for the calibration:

1. For each wheel, test what motor value can make the wheel start to rotate. The following are the corresponding values:

	Front Left	Front Right	Back Left	Back Right
Forward Rotate	28	29	39	22
Backward Rotate	-26	-29	-37	-17

Table 1: motor Values that make wheels start to rotate

2. Find a set of motor values so the robot can move forward/backward straight. The following are the corresponding values:
   - Moving Forward: Front Left: 80 – Front Right: 72 – Back Left: 80 – Back Right: 72
   - Moving Backward: Front Left: -80 – Front Right: -72 – Back Left: -80 – Back Right: -72
3. Measure the velocities when the robot moves forward/backward straight. From our measurements, they are 0.3 m/s and -0.3 m/s. Since each wheel has a radius of 3 cm, each wheel's angular velocity is $\omega = \frac{0.3\ m/s}{0.03\ m} = 10\ rad/s$ and $\omega = \frac{-0.3\ m/s}{0.03\ m} = -10\ rad/s$
4. Now, we can come up with 4 piecewise linear functions for each wheel using linear regression.

2.1 Front Left Wheel

$$\omega_{fl} = \left\{ \begin{array}{lr} 0.19231c - 5.38462 & \text{if } c \geq 28 \\ 0.18519c + 4.81481 & \text{if } c \leq -26 \\ 0 & ELSE \end{array} \right\} \tag{1}$$

2.2 Front Right Wheel

$$\omega_{fr} = \begin{Bmatrix} 0.23256c - 6.74419 & \text{if } c \geq 29 \\ 0.23256c + 6.74419 & \text{if } c \leq -29 \\ 0 & ELSE \end{Bmatrix} \tag{2}$$

2.3 Back Left Wheel

$$\omega_{bl} = \begin{Bmatrix} 0.24390c - 9.51220 & \text{if } c \geq 39 \\ 0.23256c + 8.60465 & \text{if } c \leq -37 \\ 0 & ELSE \end{Bmatrix} \tag{3}$$

2.4 Back Right Wheel

$$\omega_{br} = \begin{Bmatrix} 0.20000c - 4.40000 & \text{if } c \geq 22 \\ 0.18182c + 3.09091 & \text{if } c \leq -17 \\ 0 & ELSE \end{Bmatrix} \tag{4}$$

3 Kinematic Model

We used the basic kinematic model for our project:

$$\begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} & -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

where w_i are the wheels' angular velocities $i \in \{1, 2, 3, 4\}$,
r is the radius of the wheels,
l_x is the half of the distance between front wheels,
l_y is the half of the distance between front wheels and rear wheels,
v_x, v_y are the linear velocity of the robot in the direction of x and y,
w_z is the angular velocity of the robot.

We measured the robot's parameters and got $l_x = 6.75cm, l_y = 5.5cm$ and $r = 3cm$. To make the robot move straight, we set a uniform value for w_i at 10 rad/s, and the calculated forward speed was 0.3 m/s, which matched the measured speed. However, there were some errors when testing the rotation speed. The calculated rotation speed was 2.45 rad/s, but the robot experienced over-rotation when this speed was applied. The measured rotation speed was actually 2.06 rad/s for clockwise rotation and 2.026 rad/s for counterclockwise rotation. We used the measured rotation speed for HW1.

4 Navigation Algorithm

To move between 2 states($(x_i, y_i, \theta_i) \rightarrow (x_j, y_j, \theta_j)$), we simplify it into a 3-step procedure:

1. Rotate the robot from θ_i to $arctan(\frac{y_j - y_i}{x_j - x_i})$ – First Rotation
2. Move forward from (x_i, y_i) to (x_j, y_j)

3. Rotate the robot from $arctan(\frac{y_j - y_i}{x_j - x_i})$ to θ_j – Second Rotation

```python
def angle_process(start_angle, end_angle, no_rotation_range):
    angle_diff = end_angle - start_angle
    angle_diff = (angle_diff + math.pi) % (2 * math.pi) - math.pi
    if (-no_rotation_range <= angle_diff <= no_rotation_range):
        rotate_angle = angle_diff
        direction = 'no'
    if angle_diff < -no_rotation_range:
        rotate_angle = -angle_diff
        direction = 'clockwise'
    else:
        rotate_angle = angle_diff
        direction = 'counterclockwise'
    return rotate_angle, direction
```

Function `angle_process` calculates how to rotate from `start_angle` to `end_angle` with the least amount of rotation, it returns the rotation angle and direction. Because the robot is not always precise, we can set a `no_rotation_range` so that if the rotation angle is so small, the robot will not rotate. The function is used for step 1 and 3.

```python
def get_movement_info(x0, y0, theta0, x1, y1, theta1, no_rotation_range, scaling_factor):
    dx = x1 - x0
    dy = y1 - y0
    distance = math.sqrt((dx ** 2) + (dy ** 2)) * scaling_factor
    target_angle = math.atan2(dy, dx)
    first_rotation_angle, first_rotation_direction = angle_process(theta0, target_angle, no_rotation_range)
    assert 0<= first_rotation_angle <= 2 * math.pi
    second_rotation_angle, second_rotation_direction = angle_process(target_angle, theta1, no_rotation_range)
    assert 0<= second_rotation_angle <= 2 * math.pi
    return distance, first_rotation_angle, first_rotation_direction, second_rotation_angle, second_rotation_direction
```

Given (x_i, y_i, θ_i) and (x_j, y_j, θ_j), function `get_movement_info` returns the rotation angle and direction of the first and the second rotation and the distance between (x_i, y_i) and (x_j, y_j). We can also set a `scaling_factor` to scale up/down the distance.

Given the angles and directions of the first and second rotations, along with the forward distance, we can determine the direction and execution time for each of the four motors based on the velocities mentioned in the kinematic model (forward: 0.3 m/s, backward: 0.3 m/s, clockwise: 2.06 rad/s, counterclockwise: 2.026 rad/s).

5 Code

We submitted the zipped `rb5_ros2_control` folder. You may find the following code files in path `rb5_ros2_control/rb5_ros2_control`:

- `wheel_start_test.py`: Measure the motor values that make wheels start to rotate.
- `forward_calibration.py`: Given a set of motor values that can make the robot move forward straight, measure the robot's moving velocity.

- `backward_calibration.py`: Given a set of motor values that can make the robot move backward straight, measure the robot's moving velocity.
- `clockwise_rotation_calibration.py`: Measure the robot's clockwise rotation angular velocity.
- `counterclockwise_rotation_calibration.py`: Measure the robot's counterclockwise rotation angular velocity.
- `hw1_code_version1.py`: Run this file for HW1.

6 Comments

The algorithm allows the robot to traverse between different waypoints. However, it deviates slightly for point (-2, 1, 0) and (-2, -2, -1.57). The deviation should be less than 10 cm.
The errors may come from the following aspects:

- Inaccurate measurement of robot rotation angular speeds and moving forward/backward speed.
- Motors may have different performance for different battery levels.

The following are possible improvements:

- We can measure more points for calibration.
- Currently, we set 4 motor values as hyperparameters and they do not change while the robot is moving. Later, we can design an algorithm to give different motor values to different wheels at different locations. This can allow the robot to traverse the waypoints more efficiently.