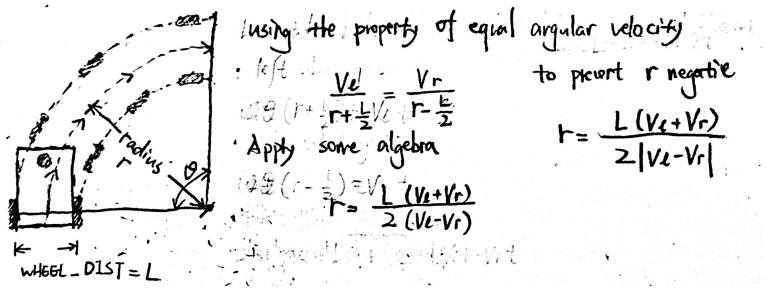
CS 3630 Project 3 Part 1

Name: Fei Ding
GT username: fding33
GTID: 903444656
Group #: 19

Name: Zhen Jiang
GT username: zjiang330
GTID: 903402987

1. How did you implement calculate_radius?

First we need to detect the edge case, namely when the left wheel velocity is equal to the right wheel velocity. In this case, the robot is expected to perform pure translational movement forward, for which we particularly decide the radius to be 0. Then we derived the formula for the radius given left and right wheel velocity using my knowledge about the differential drive robot (Shown below). The mathematical formula is easy to translate into code. Since the radius can become negative in accounting for the direction of turn, we will take the absolute value of the result before returning.



2. Screenshot and paste the commands returned when you run command rosrun project3 move_in_circle.py

```
root@duckie019:/code/catkin_ws/src/project# rosrun project3 move_in_circle.py
[INFO] [1581440814.221201]: [/move_in_circle] Initializing...
[INFO] [1581440814.235778]: Publishing message: 'Hello from duckie019'
[INFO] [1581440814.237070]: Publishing message: Moving in radius: '0.112200'
^C[INFO] [1581440821.666071]: [/move_in_circle] Shutdown.
```

3. What are the initial velocities you used? What is the returned radius? How did the robot behave when you ran it with those configurations?

The initial velocity settings we used were {“velocity_left”: 0.3, “velocity_right”: 0.8}, The returned radius, as calculated by the function we implemented, is 0.112200. The robot did move in a circle counter-clockwisely. The radius of the circle it moved in was close to what we expect, but not precise enough. The speed of movement was hard to track but a little bit calculation in the frequency of its periodic movement suggests it’s not too far off. However, we observed that the robot drifted on the floor slowly. In other words, the circle was not perfect, so the robot would astray to other places rather than staying in the circle where it should be.

4. Try making the robot move in a ‘better’ circle. (TIP: try adjusting the wheel speeds and/or the radius. Running on different surfaces may return different results.) What did you do to make the robot run in a circle of similar radius as calculated? Write the velocities and calculated radius you used.

In order to make the robot move in a better circle, we tried to have the robot move on different surfaces and use different velocity settings. Surprisingly, we found the carpet was too “slippery” for the robot, but the tiled ground is actually friendly to it. We experimented with different velocities. A trivial one would be making the left and right velocities opposite, for which our robot just spinned in one place precisely with little deviations, but this is not practical in all settings as we do need to make the robot turn smoothly. We found that a small radius (~0.1) and medium velocity (avg(velocity_left, velocity_right) ~ 0.5) usually produce a good result. The setting we used was {“velocity_left”: 0.3, “velocity_right”: 0.8, “radius”: 0.10}.

5. Why do you think the robot did not move as you thought? What can you do (Hardware and Software) to improve the results?

The floor texture and speed matters: if the floor is slippery, there is no chance of it moving correctly, especially true with a fast-moving robot. There is also problem when moving too slow: the motor is only manufactured to a certain precision, so a small deviation in motion may add up to give a far-off trajectory. To make the robot moves more smoothly, we decide to have the robot drive on frictionless surfaces like wood. We also try to move with optimal speed (neither too fast nor too slow). We can also improve on the motor (unlikely in our case). Speaking on software, if the robot drifts consistently left or right, we can to set a trim value for the tires. If we are not “blind” and can observe the environment, we can make a sophisticated mechanism for the robot to adjust its motion as it detects a deviation from its plan.