CMPUT 312 lab1

Group members: Sergey Khlynovskiy, Jerrica Yang

1)

We did not iterate on our designs and stuck to the first robot that we made because it did the job well enough. We focused on limiting parts usage to create an efficient design. One thing that could have been improved upon is moving the gyroscope away from the middle of the computer so that it does not block the screen, but simply taking that part on and off was not too troublesome.

Find the photos in the media folder.

2) Error collection and analysis – class used: ev3dev2.motor.MoveDifferential

- Linear:
  Method 1: comparing set distance in the program vs actual distance traveled

  Method 2: using the encodings of the wheels by calculating the amount of rotations * the circumference of the wheel.

Comparison of the two methods using function on_for_dirstance(speed_percent, distance)

Comparing the measurements with the two methods, the robot seems to overshoot the target distance slightly at all speeds, and this effect becomes more significant at higher speeds. Since the encodings of the wheels were the most accurate at predicting actual distance traveled, we performed linear regression on the 4 data points and found an equation for the error of wheel encodings to distance traveled (e) as a function of wheel speed percentage (v). e=0.09v+0.2. Additionally, higher speeds introduce some difference between the left and right wheel encodings, indicating potential drift or unbalanced movement.

- Rotational:
  Method 1: Comparing initial and final gyro angle
  Method 2: using the wheel encoding to find the change in orientation [1] by doing

$$\Delta\Theta = \frac{D_R - D_L}{L}$$

  change in orientation (Δθ), measured in radians, is influenced by the difference in wheel displacements and the distance between the wheels (L). In our case L = 198 mm[1]

Comparison of the two methods using function turn_degrees(speed=speed_percent, degrees= 90, error_margin=2, use_gyro=True)

The real angle seems fairly close with the target angle. The gyro sensor gives a more accurate measure of the actual turn angle compared to the wheel encoders. The calculated angle from the wheel encoders consistently reports a much larger turning angle, which suggests that

relying on encoders alone might overestimate the turn angle. The cause of overestimating might be from the wheels jittering or slippage during rotation. The small movements might cause an additional length added to the distance each time.

Performing linear regression on the 4 points using the gyro angle method we find an equation for error of actual angle - gyro angle (e) as a function of wheel speed percentage (v).
$e=0.2v+0.45$.

note: we used the differential drive class for performing the rotations for this part as well as part 3 and we found that we needed to adjust the wheel distance significantly from what it actually was (198mm). When it was at 198 it would overcompensate and make big left to right turns before aligning itself at the appropriate angle. When the angle was < 100 it would jitter to get into position though at 100mm it would sometimes miss the turning angle. We found that 50mm was the most consistent from our testing and is what we used.

3) class used: ev3dev2.motor.MoveDifferential

- Rectangle:

  Conclusion:

  The gyro sensor provided accurate data, but the wheels introduced errors when turning in a 90 degrees angle, resulting in a 20 mm difference between the starting and ending positions. This difference was likely caused by slippage and wheel/ surface friction during turns. By recalibrating the gyro sensor at the start only and using slower speeds for turns, the error was significantly reduced. While travel distances in straight lines were consistent, a slight drift of 0-2 degrees occurred. Overall, the adjustments helped improve accuracy.

  Error analysis:

  Based on the analysis from part 2, we observed that the gyro sensor provided data closely aligned with the expected target values, demonstrating its reliability. However, the calculated angle using the wheel encoders and the change in rotation formula shows a bit of error. Theoretically, the robot's starting and ending positions should be nearly identical, yet we observed a difference between start and end point of around 20 mm, and the first straight line in iteration 1 differs by 6 degrees in iteration 3, which suggests cumulative errors over time.

  Since the robot is navigating in a rectangular path, the total gyro angle and calculated angle should ideally sum to 360 degrees. However, small deviations suggest some degree of slippage with each turn, which becomes more evident when we analyze the change in orientation. Our data shows that as the robot

continues to turn, the motor's wheels traveled distance becomes a major factor of the error, possibly due to slippage, wheel friction, or other mechanical factors.

We tried to compensate for the errors each turn by remembering the previous angle but that didn't work so well because we recalibrated after each turn. Rather than trying without recalibrating and compensating for the previous angle, we switched to recalibrating once at the beginning of iteration 1. It worked best after trial and error using self.turn_degrees(

```
        speed=SpeedPercent(15),
        degrees= deg,
        error_margin=2,
        use_gyro=True,
    )
```

especially on the table as that eliminated slippage.

As for the linear portions of the rectangle, the travel distance collected was fairly consistent compared to manual measurement, likely because we set the speed percentage to a low value (from part 2 linear analysis). However, one error we observed was a slight drift of 0-2 degrees while driving in a straight line. This drift could be attributed to minor differences in motor performance, wheel wear, or surface frictions, causing the robot to drift. Over long distances, even small drifts can accumulate and affect the overall accuracy of the robot's trajectory.

The actual distance traveled was 246mm and the calculated encoder distance was 248.7mm so a difference of 2.7mm, so quite close. We told the robot to go for 150mm which is off from the actual distance which coincides with what we found in part 2.

The gyro angle tells us that the robot turned 88 degrees and the actual turn was 91, a difference of 3 degrees. Using the encoding of the wheels we find that the robot turned (-176.4-156.6)/198=96 degrees. Overshooting which is what we found in part 2.

- Lemniscate

  Conclusion:

  We performed the lemniscate by completing two adjacent circles. To create the circle we used the on_arc_left/right functions rather than the go function which gave us a radius to compare with the actual drawn radius. A while loop would continue until the angle was within 10 degrees of a completed circle. This was to ensure that the robot didn't overshoot or undershoot too much. For a smoother circle we could have used the go method for a more continuous turn.

  Error analysis:

In this experiment, we observed an 80 mm difference between the start and end positions of the three lemniscates. This error seemed to be from inaccuracies in the distance covered during each half of the pattern. After multiple runs, the drawn figure drifted downward, indicating a cumulative deviation that grew with each iteration.

The radius of each circle is roughly 400mm in diameter. This fits as in the code we did self.on_arc_right(
speed=SpeedRPM(30),
radius_mm=200,
distance_mm=30).

We incrementally performed the circle so we can use the gyroscope to know when we completed a full rotation. The gyro angle was set to slightly less than what it would need to complete the turn as we could overshoot the angle otherwise because we are not counting the angle mid-movement.

The downward movement of the lemniscate can be attributed to a non perfectly symmetrical build of the robot. This could be a similar effect as was observed when driving in a straight line and the robot would slightly drift.

Since there are no straight lines or perfect turns we cannot use the methods from part 2 directly, however, using the encoders we find that the robot traveled 1289.7mm on the first circle which is quite close to the actual circumference of the circle (400*pi)=1257 mm. Also, we can compare gyro angle and actual angle after each circle. Doing so for the first circle in the first iteration we find the end angle to be 1 degree and the gyro angle measured 365 degrees so an overshoot by 5 degrees. A difference of 4 degrees.

4) class used: ev3dev2.motor.MoveTank(

We went back to MoveTank class for the movement here due to the reduced complexity of setting wheel distance as was mentioned in part 2. Though, this did have some drawbacks in performance. For example when performing turn_degrees it would overshoot the angle more than was observed with the MoveDifferential class.

Speed Measurement: We ran both motors at full speed for 1 second using on_for_seconds(SpeedPercent(100), SpeedPercent(100), 1), which gave us a speed of 48 cm/sec.

Rotation Calculation: We found the wheel's rotations per second based on its circumference and distance traveled in 1 second, resulting in an angular velocity of 2.18 rotations/sec. From there, we calculated the speed for each wheel based on the speed percentage at a specific time. Note that we assumed both wheels had the same performance, no matter the orientation of the vehicle.

Angular velocity of the whole motor (w):

$$\omega = \frac{v_r - v_l}{2d},$$

2d is the length of mid wheel left to mid wheel right, in our case was 198mm

Rotation radius (R):

$$R = d\left(\frac{v_r + v_l}{v_r - v_l}\right)$$

Velocity of the whole motor (v): $v = R * \omega$

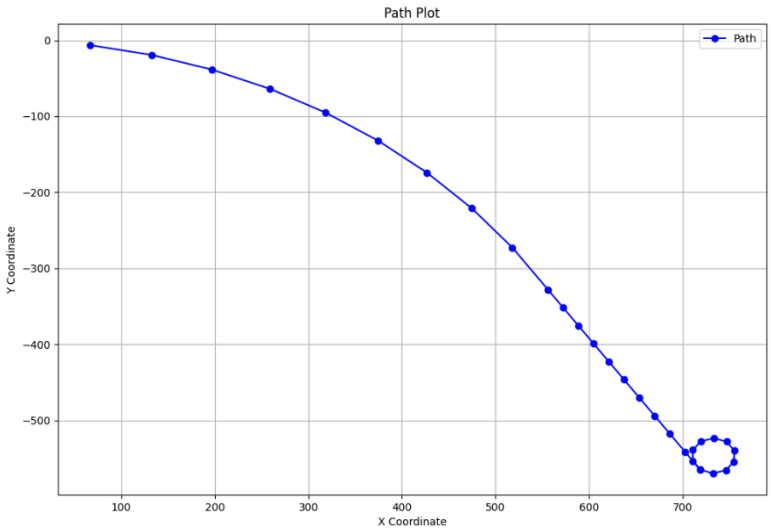
Location (x, y, theta):

$$x(t) = \int_0^t v(t)cos(\theta_t)\, dt$$

$$y(t) = \int_0^t v(t)sin(\theta_t)\, dt$$

$$\theta(t) = \int_0^t \omega(t)\, dt$$

The values were calculated by constant velocity for each command, and performed a left riemann sum with 10 steps to calculate the integrals. Resolving in a theoretical graph of the trajectory of the robot (graph x).



Graph x: theoretical graph of the trajectory of the robot

Error analysis:

The maximum rotations per second that we set as a constant in our program is actually heavily affected by the surface that the robot is on as well as the type of maneuver the robot is trying to perform.

Using the error calculating methods from part 2 we can find the linear error and the rotational error for parts of the movement. Starting with the linear and taking as example the first run, we see that when the robot went in a straight line the wheels moved from 972.5mm->1348.7mm for the left and 756.3mm->1125.2mm for the right. This suggests that the robot moved roughly 372.6mm in a straight line. The actual measured straight line can be found using pythagoras theorem and is roughly 329mm, a difference of 44mm. For the rotation we can also look at the gyroscope and see that it is quite close to what we measured. Taking the first run again as an example, the gyro angle and the true measured angle are within 10 degrees. We won't use the wheel encodings to measure the angle error as that was shown to be quite off in part 2.

Now, comparing the estimated position and orientation from the dead reckoning is where we find more of an error. The measured and calculated locations differ by roughly 100 - 200 mm, suggesting a deviation between theoretical and actual movement. In all three rounds of linear movement (command 2, table x), the robot consistently traveled approximately 10 cm further than predicted. This indicates that the wheels may have been over-rotating, causing the robot to overshoot its theoretical distance. This was also seen in part 2, where we showed that higher speed gives larger error, and the actual distance given will be a lot more than our theoretical travel distance.

Round	Measured x	Measured y	Actual distance (mm)	Theoretical travel distance(mm)
Iteration 1	870	-575	329	288
Iteration 2	900	-620	402	288
Iteration 3	820	-600	340	288

Table x: comparing between the command 2 over all three iterations

Other reasons that we think might be a cause of the error is the wide length between wheels (2d) we have for the motor. We noticed that in the third command (-50, 80, 2), we resulted in a R value smaller than the d. This means that the robot will be facing a sharp turn. The angular velocity is affected by the length (d) from the formula

$$\omega = \frac{v_r - v_l}{2d},$$

and the slight difference between the left and right difference of wheel velocity may be a cause of the error as well. Here we are assuming that both the wheels have the same speed when at

max performance. Despite the trajectory seems to be fairly similar to our theoretical one, wheel slippage and surface friction are still contributing to deviations in the robot's actual movement (seen in part 2 and 3). Also, changing into MoveTank class has increased the error (mentioned above). These factors introduce errors in the robot's motion, affecting both the accuracy of the distance traveled, location calculation and lowers the accuracy of each turn.