Closed-Loop Controller: Driving a mobile robot straight

Zouhair Adam Hamaimou
1004891986
zouhair.hamaimou@mail.utoronto.ca

***Abstract*—This report aims to summarize the key findings from completing the task of creating a closed-loop controller for a rover.**

Keywords—error, control, PID

I. SETTING UP

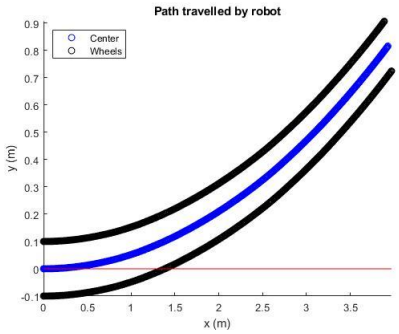


Figure 1: Unadjusted simulated path of rover

The deviation of the robot from the intended path in Figure 1 is due to:

- Uneven surfaces (bumps and obstacles) simulated in newPos.m by multiplying the intended changes in the motors (dL and dR) with a random normally distributed noise computed in mynoise.m.
- The asymmetry of the motors which is simulated by making the left motor 2% stronger in newPos.m. This causes the right motor to be slightly faster leading to a deviation to the left of the intended path.

The number of seconds was adjusted to N = 100s. This provides a more complete picture of where the rover is headed (as the settling could occur later). It also maintains the time to run the program at a reasonable duration (for my PC which is not the fastest).

The resolution was kept at pps = 1000Hz. By trying different numbers, I found that increasing the resolution does not make the trajectory any more accurate but simply makes the run time longer. Decreasing it below 1000Hz changes the trajectory. Therefore, 1000Hz is the lower bound to discretizing the differential equation for the motion of the rover in getPos.m.

I will choose the same rover specifications that were used for the rover in workshop because I already have a sense of this rover's movement and I will be able to apply this project to it (since our team kept our rover).

- The wheels from the MYLFF have a diameter of 65mm., so r = 32.5mm
- The motors used in Workshop have a maximum angular velocity ω_{MAX} = 100rpm = 1.67 rots/sec
- The distance between the wheels is approximately the width of the board used in Workshop, d = 5.75" = 0.146 m

Spd is the initial speed of the robot in meters per second and is used as an initial condition from which the system evolves according to the difference equations we set. We need to make sure it does not exceed the max speed allowed by motors.

dT is the time interval between controller changes. It is used to get the control frequency which is the inverse of dT. We divide this control frequency into the simulation frequency pps to find the control interval. This is the number of intervals between deviation checks and is 10 points in our case. Therefore, pps must be an integer multiple of 1/dT in order to avoid float values in the deviation check for loop.

Ctrl_enable is a value to choose which control scheme to use which can be any combination of proportional, integral, and derivative control. So far, we have two cases, 0 for no control and 1 for proportional control, but more cases will be added in Part III.

The function getDC:

- converts the desired linear velocity to an angular velocity ω by dividing it by the perimeter of the wheel which is 2πr
- finds the duty cycle by computing: $\alpha = \omega / \omega_{MAX}$
- makes sure the duty cycle does not exceed 1

II. CLOSED-LOOP PROPORTIONAL CONTROL

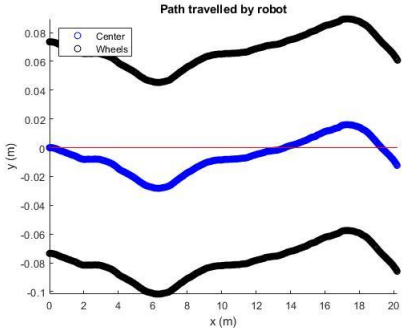
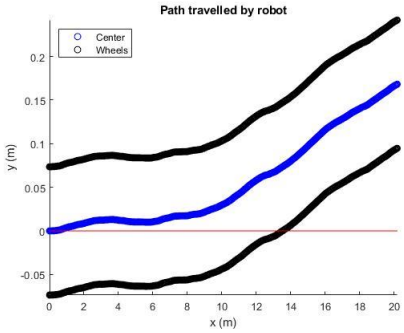


Figure 2: Path adjusted using proportional control

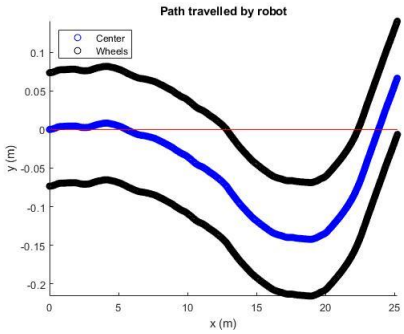
After switching ctrl_enable to 1 and setting the proportional parameter Kp = 150, the trajectory of the robot has changed to show oscillations as shown in Figure 2. This is done by checking, after each control interval, the encoder error between the left and right motor and adjusting the motors' speeds by a factor Kp of the calculated error.

In getError, we first simulate the change in both motor's encoders over the control interval. The difference gives us an estimate of how much one motor got ahead of the other. However, since the encoders do not accurately reflect how much the motor has truly rotated due to noise, they are not sufficient to calculate the true error ($\epsilon = x - x_D$). According to [1], a gyroscope would be more helpful in detecting deviations from intended path because it records processed information about the direction the robot is headed in.

Setting the proportional parameter Kp too low as in Figure 3(a), may result in not enough of a correction to the robot's direction. While setting Kp too high as in Figure 3(b), may result in an overshoot of the correction making the robot head in the opposite of deviated path. Therefore we need to find the right Kp that will take the robot exactly back to its path. This process consists of repeated trials and errors.



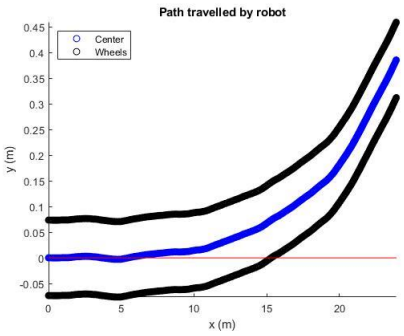
(a) Kp = 75



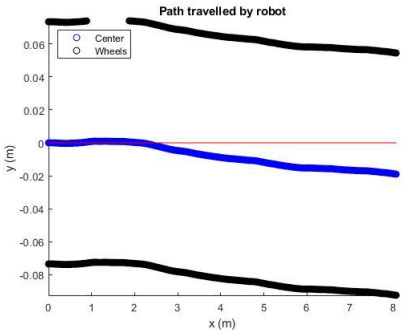
(b) Kp = 200

Figure 3: Adjusted path with different proportional parameters

Increasing dT leads to a larger control interval, which may lead to missing a correction when needed. In Figure 4(a) we see that increasing dT by a mere 0.05 will lead to a great deviation (this happens regardless of Kp). While decreasing dT, leads to a smaller control interval making corrections more frequent and leading to a straighter trajectory but requires the parameter to be tuned again. However, there is a lower bound on dT as the control frequency 1/dT cannot exceed the simulation frequency pps. Also due to processor capabilities, notably in smaller robots with smaller microcontrollers. In Figure 4(b), dT was decreased to 0.001 so pps had to be increased to 2000Hz to make up for it. Nonetheless, a straighter path is noticeable.



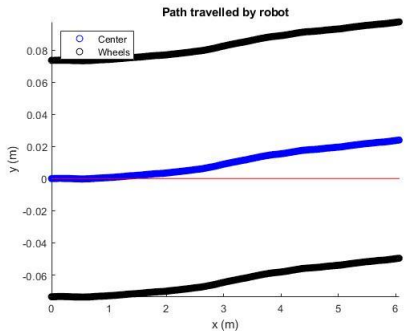
(a) dT = 0.015



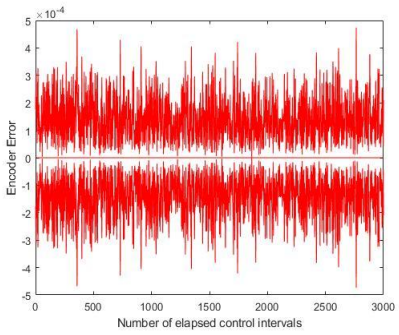
(b) dT = 0.001

Figure 4: Adjusted path with different controller frequencies

In order to analyze the error, we create an array of encoder errors and plot it. By visually comparing the error plot and the trajectory side in Figure 5, we notice that the error is typically within a range [-0.0003,+0.0003] but certain peaks that exceed that range. Notably, the peak around the 400th control interval exceeds the typical error range and could not be corrected with our current value of Kp. This leads the robot astray.



(a) Kp = 75



(b) Kp = 200

Figure 5: Encoder error analysis

The time taken to reach the dashed red line in Figure 6 is the settling time of our controller. It is defined in [2] as the time required for the signal to stay within a chosen tolerance of the final signal.

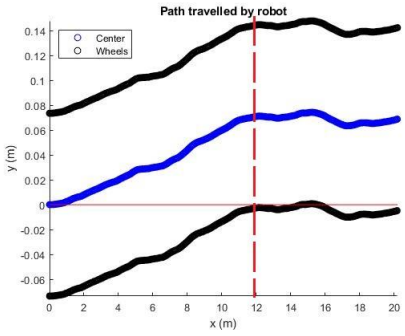


Figure 6: Settling Time

The length of the red segment in Figure 7 is the overshoot, which is defined as the maximum deviation from the final signal.

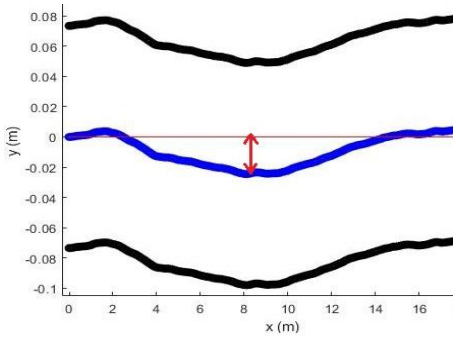


Figure 7: Overshoot

III. Closed Loop Control

The proportional part of the PID has been completed.

Since we only compute values of the error after each control interval, we use dT as our time interval for the integral and derivative portion of the PID.

For the integral part, we turn back to Riemann sums as a discrete approach to computing the integral of the error. For greater accuracy, we use the trapezoidal approximation: beginning at I = 0, after each control interval we add to I the area under the trapeze formed by the bases being the previous error and current error and height dT.

For the derivative part, similarly we will need to compute a ratio of differences instead of a derivative. At each control interval, we divide the difference between the current and previous error by dT to obtain an estimate of the derivative.

We use the famous Ziegler-Nichols method [3] to tune the PID parameters. It consists of the following steps:

1- Set all parameters to zero.

2- Increase Kp until the response to a disturbance is steady oscillation. (Already done in Part II)

3- Increase Kd until we reach a critically damped system with no oscillations

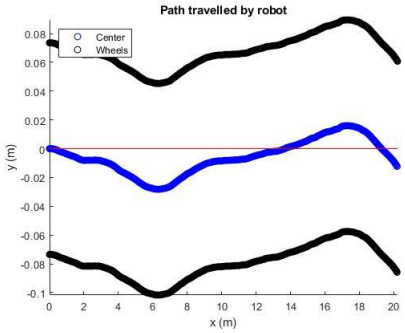
4- Repeat steps 2 and 3 until we reach a point where increasing Kd fails to make the system critically damped.

5- Set Kp and Kd to the last stable values.

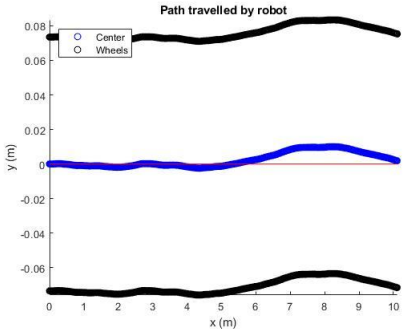
6- Increase Ki until the desired number of oscillations is reached(usually none but settling time can be decreased if a couple of overshooting oscillations are ignored)

If the oscillations start increasing, then we reduce Kp.

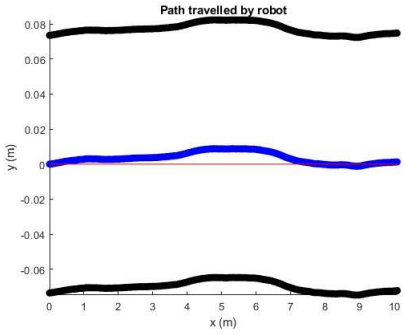
If Kd is set too high, the system will begin to oscillate at a frequency higher than the proportional control done in Part II. When this happens, we reduce the Kd until these vibrations are eliminated.



(a) Kp set (150)



(b) Kd set (0.08)



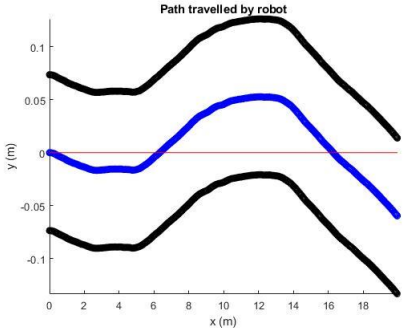
(c) Ki set (3000)

Figure 8: Tuning of PID parameters using Ziegler-Nichols

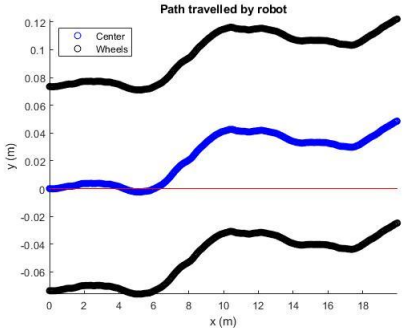
Figure 8 shows how the parameters were tuned following the procedure mentioned above. We see that this approach is appropriate since it makes adequate use of each portion PID control.

The proportional control only sees current error and can never completely remove oscillations leaving us with an

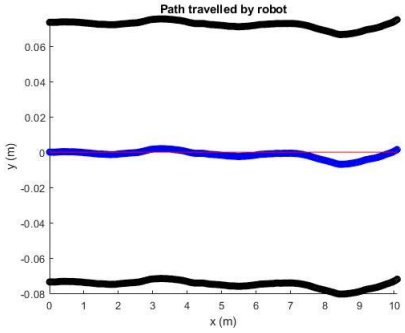
underdamped system in Figure 8(a). Then, comes the derivative control that predicts future error using rates of change to prevent oscillations from occurring. However, it does not consider the accumulated error and leaves us with an overdamped system in Figure 8(b). Finally, the integral control considers past errors and integrates them in order to take the system to a critically damped state in Figure 8(c).



(a) Only P control



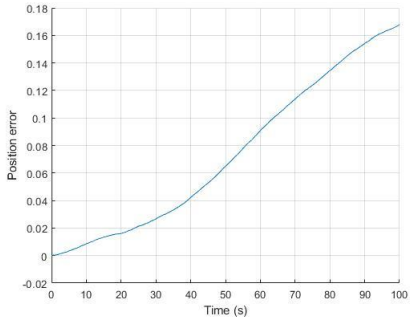
(b) PI control



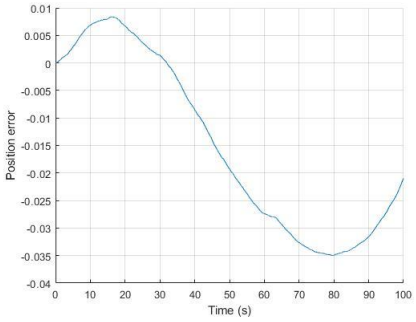
(c) PID control

Figure 9: Comparison of different control schemes

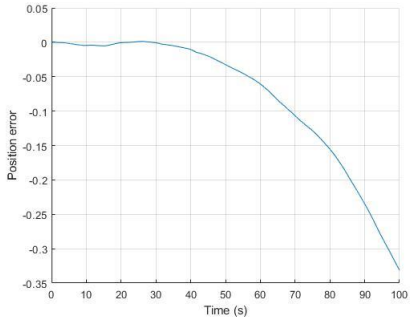
In Figure9, we compare different control schemes. It is clear to see that with proportional control (a) settling is very difficult because we are not accounting for accumulated error and not attempting to predict future error. On the other hand, PI control (b) manages to go back to being straight upon deviation, after noticing the accumulated error. Finally, PID control (c) prevents that initial deviation from occurring at all by predicting it and correcting with a term proportional to the derivative of the error.



(a) K = [100, 2500, 0.06]



(b) K = [150, 3000, 0.08]

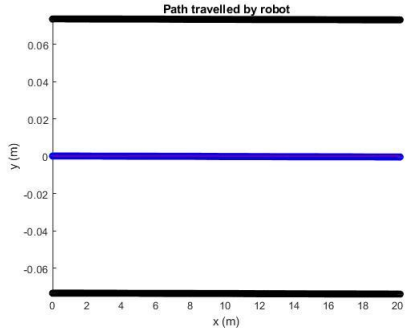


(c) K = [200, 3500, 0.10]

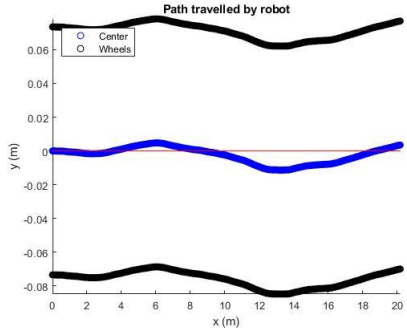
Figure 10: Position-error vs. Time

In Figure 10, we see how the position-error as a function of time varies with different parameter vectors [Kp, Ki, Kd]. For the correct parameters (b), the position-error should oscillate around 0. In cases where the parameter is too small (a) or too large (c), the position-error accumulates in one direction and does not return to the stable point.

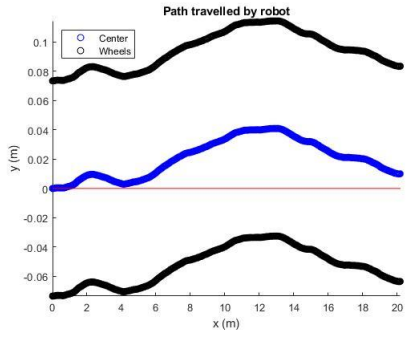
PID controls have many advantages, mainly they are not too complicated and serve their intended purpose. However, they are very sensitive to measurement noise. The simulated error of encoder measurements caused this task to be much more complicated. In Figure 11, we plot the best result over several trials of the adjusted path with tuned parameters for encoders with different variances σ^2. For an ideal encoder (a), the PID controller performs perfectly. For the encoder we were given (b), the PID still manages to adjust the path. However, even for a slightly less accurate encoder (c) we see that the PID is not able to adjust the path as the error input is given is simply too inaccurate.



(a) $\sigma^2 = 0$



(b) $\sigma^2 = 0.0001$



(c) $\sigma^2 = 0.0005$

Figure 11: PID with different encoder noise

The analysis done of Figure 11 leads us to conclude that for our rover that has DC motors equipped with encoders of a wide range of accuracy, it will prove very challenging to use PID in order to drive the rover straight. If somehow, we are able to find more accurate encoders or use a different sensor that provides more accurate error measurements (such as a gyroscope), we can rely on PID to drive the rover straight.

Furthermore, the proportional parameter of the PID controller is constant. While the integral and derivative parameters depend on time, they can be heavily impacted by an error that has already been corrected. More flexible control parameter that are readjusted at a certain interval would reduce deviations and lead the system to the intended path at a lower settling time and with minimum overshoot.

IV. Conclusion

The PID controller can be implemented to our rover but will require a more precise error measurement device and the implementation of a system that can correctly retune the control parameters when needed.

References

[1] Park, J.-H., & Cho, B.-K. (2018). Development of a self-balancing robot with a control moment gyroscope. International Journal of Advanced Robotic Systems.

[2] Joe W. Yeol, Richard W. Longman, Yeong S. Ryu (2008). On the Settling Time in Repetitive Control Systems, IFAC Proceedings Volumes, Volume 41, Issue 2, Pages 12460-12467

[3] Åström, K. J., & Hägglund, T. (2004). Revisiting the Ziegler–Nichols step response method for PID control. Journal of process control, 14(6), 635-650.

[4] Bhatti SA, Malik SA, Daraz A. Comparison of PI and IP controller by using Ziegler-Nichols tuning method for speed control of DC motor. In intelligent systems engineering (ICISE), 2016 international conference on 2016 (pp. 330-4).

[5] Singh AP, Narayan U, Verma A. Speed Control of DC Motor using Pid controller based on matlab. Innovative Systems Design and Engineering. 2013;4(6):22-8.

[6] Meshram PM, Kanojiya RG. Tuning of PID controller using Ziegler-Nichols method for speed control of DC motor. In advances in engineering, science and management (ICAESM), 2012 international conference on 2012 (pp. 117-22).

[7] Ahmed A, Mohan Y, Chauhan A, Sharma P. Comparative study of speed control of DC motor using PI, IP, and fuzzy controller. International Journal of Advanced Research in Computer

[8] Robert Habib Istepanian. (1997) Implementation Issues for Discrete PID Algorithms Using Shift and Delta Operators Parameterizations. IFAC Proceedings Volume 30, Issue 3 (pp. 113-18).