Table of Contents – Closed-Loop Controller

Part 1

Question 2

Some potential factors that can cause the trajectory of the robot to deviate is discrepancies in the strength between the motors. If the left motor has a slightly greater duty cycle than the right motor this can cause the trajectory to deviate as well as a particularly rough surface.

The positional disturbances are shown in the my_noise and my_noise2 function. The my_noise function is used for rough surfaces and obstacles while my_noise2 is used to generate random noise readings from the encoders. Both functions are then called in the newpos function to simulate robot movement.

The motor power offset is shown in the code below:

```
lDC = robot.lMot*1.002; % Left and right motor duty cycle | left motor is 0.2% stronger
rDC = robot.rMot;
```

Question 3

I changed the value of N from 40 to 20 for the simulation time of the robot. This value was chosen because it is the approximate time spent aligning the robot along one axis of movement. The project guidelines indicate that we are allotted 120 seconds, and given the robot will complete about 5 additional tasks, this time was divided by 5 in order to obtain a value of 20 seconds per task.

I changed the PPS value from 1000 to 2000 and this was done by considering the speed and accuracy of the simulation. If a PPS value is chosen to be too high, the time it takes to generate the robot trajectory will be too long. If a PPS value is chosen to be too low, the accuracy of the robot trajectory will be insufficient.

Question 4

R was chosen to be 0.05 m which is approximately the size of the wheels used in the rover for the assignment. wMax was chosen as 1.16 which is approximately the maximum rpm of the DC motor (70 rpm) used for the assignment. Finally, d was chosen as 38 cm which is the distance between the two wheels used for the assignment.

Question 5

- Spd (main_script.m) → robot.setSpeed (main_script.m) → getDC function (drive_robot.m) → converts desired velocity to the specific duty cycle (getDC.m) → the duty cycle is then used to determine the next position the robot will be found with newPos.m (drive_robot.m)
- dT (main_script.m) → which is called in drive_robot function (main_script.m) → drive_robot function (drive_robot.m) → ctrl_interval = dT/Tstep which is used to determined is a feedback loop (causing adjustment in robot position) is necessary
- Ctrl_enable (main_script.m) → 0/1 which disables or enables PID control for modifying the loop in drive_robot.m (drive_robot.m)

dT and pps need to be matched as multiples of each other. If this is not the case, complications can arise in the simulation of the movement of the robot. In the drive_robot.m function, we see that the ctrl_interval is determined by dividing dT by the time step, which is obtained using the pps value. If they are not multiples of each other, there will be noticeable gaps in the simulation process due to the remainder caused by the quotient of these two values. These gaps will probably not be visually noticeable due to the relatively small magnitude of these values, but it could cause complications in error.

```
Tstep=T(2); % Step size of simulation
ctrl_interval=round(dT/Tstep);
```

Question 6

The function drive_robot.m first calculated the angular velocity, denote by w, required to drive a wheel with the inputted parameters of speed (v) and radius (r). The duty cycle is defined as the percentage of time needed to keep the motor on, and therefore, the time the motor is kept on corresponds to a particular angular velocity using the ratio w/w_{max}.

```
function DC = getDC(v,r,wMax)
        w = v/(2*pi*r);
        DC = w/wMax;

        if DC > 1
                DC=1;
        end
End
```

Part 2

Question 1

The value of the proportional coefficient k is 0.1 as defined in the drive_robot.m function. The ctrl_enable variable in this case is set to 1, which means the proportional control case is utilized in the code. In this code, if the deviation in the robot's x position becomes greater, the duty cycle of the motor will be adjusted by a factor of k*pos_error. This ensured that the robot will go in the opposing y direction and back to the desired path direction and move in a straight line, which is highlighted by the code shown.

```
robot.rMot = robot.rMot-K*pos_error;
robot.lMot = robot.lMot+K*pos_error;
```

Question 2

getError is used in drive_robot.m to determine the magnitude of adjustment needed to make when the robot deviates from the outlined path along the x axis, by subtracting the value of the right encoder from the left. The use of encoders is not always sufficient for accurate odometry. In short, using solely encoders in order to achieve odometry for short distances is quite accurate (which is the situation in this case), however, as the robot travels over larger distances, the error values increase quickly and causes ineffective movement. [1] Additionally, even in enclosed environments, robot wheels slip on the surface they are travelling, which can cause the encoder reading to be higher in comparison to the distance the robot has actually travelled, which can cause discrepancies in data collection. [2]

One potential sensor that could be used to optimize odometry is an Inertial Measurement Unit (IMU) which combines 3 axis gyroscopes with 3 axis accelerometers which provide a very accurate reading of angular and linear acceleration. [2] These values can then be used to extrapolate the position and direction of the robot based on its known starting position.

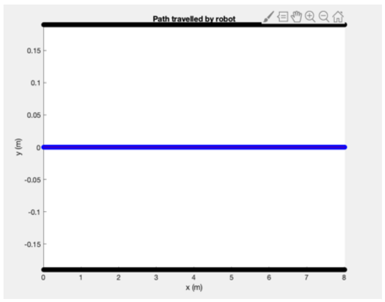


Figure 1- k=100 and dT=0.001

Question 3

When the value of k is relatively small, there is a very limited amount of control being utilized. On the other hand, when the value of k is made larger, the movement of the robot approaches the desired path as required (figure 1). When the value of k is chosen to be too large, there is a growth in the number of oscillations. This is because the correction factor is too large and overshoots the equilibrium position of the straight line. While simply using smaller values for proportional control can be accurate for straight line travel when the robot is required to make turns, the magnitude of the constant will not allow for a quick enough response to the change in direction, which can cause the robot to veer away from the desired path.

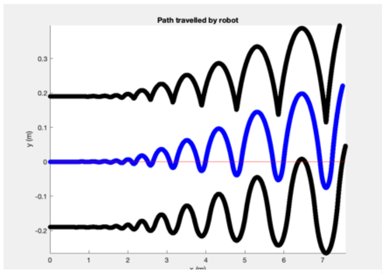


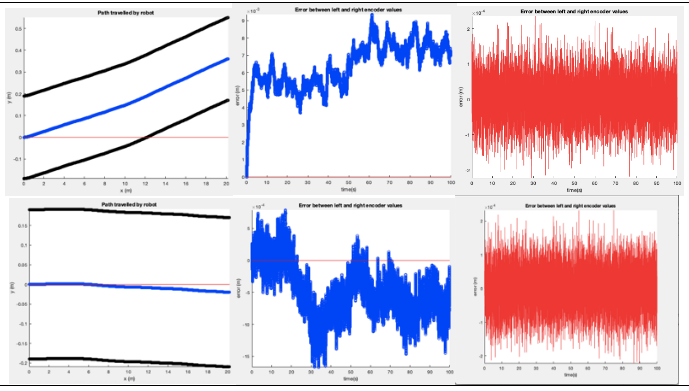
Figure 2- k=100 and dT = 0.1

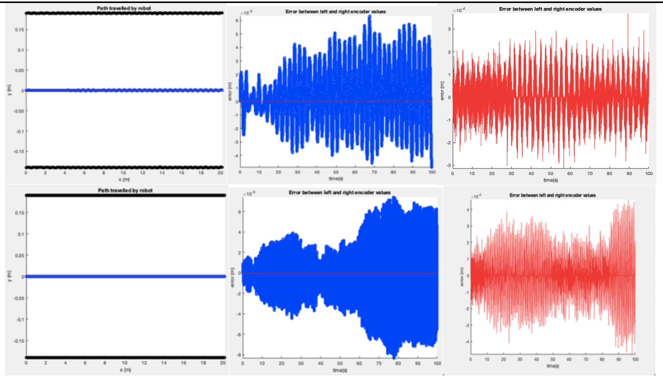
As dT gets smaller, the frequency that the PID control is initiated decreased. Changing dT affects how often the robot will attempt to correct its trajectory towards the desired path (figure 2). Overall, k determines the magnitude of the degree of the correction and dT determines how often a correction needs to be made.

Question 4

In understanding the effect of k by analyzing error feedback signals and system overshooting, the case of regular proportional control will be simulated as well as compass control (further explained later) in order to obtain more visible oscillations.

Proportional Control	**Compass Control**





Note: the blue error graph represents encL-encR versus time while the red error graph represents enc_error versus time. As shown above, the enc_error versus time graph looks relatively similar for the different cases. Graphing encL-encR displays how much further the left wheel travels at each time step in order to provide more concrete evidence for robot behaviour (the same format is used for all following cases). When the encL-encR is primarily above the x axis, it signifies the left wheel is consistently travelling a further distance than the right wheel at each time step and vice versa for below the x axis. Ideal behaviour is approximately an equal amount of time spent both below and above the x axis in order to signify the most accurate correction behaviour towards the desired pathway. Using only enc_error versus time graphs would make deciphering the associated robot trajectory quite difficult which is why the encL-encR graph was added.

The top three images above display the robot trajectory and associated encoder error (L minus R) with k=1 and the right image displays k=10. In analyzing the case on the top, the error graph shows that the robot trajectory should be veering to the right as left encoder > right encoder. This is reflected in the robot trajectory, however, without significant correction due to the relatively low k value. The figures on the bottom displays what happens when the k value is increased. As shown, the encoder error oscillates much closer to 0 (desired behaviour) as the robot's correction factor is reasonably larger and is therefore more critically damped. The error lags primarily in favour of the right encoder, which means the robot will veer in the left direction as shown. Due to the higher k value, the robot is able to more accurately correct its motion as shown by the relatively horizontal slope of the robot's trajectory at the end of the simulation.

The top three images above display the robot trajectory and associated encoder error with kc=0.03 and the bottom three images displays kc=0.3. In analyzing the case on the top, the robot trajectory oscillates relatively rapidly around 0. The kc value in this case is not high enough which causes an underdamped system, as displayed by rapid overshoot periods as the simulation progresses. This means the robot takes too long to correct its position, as evident by the small gaps between error readings. The figures on the bottom shows a critically damped system which hardly has any oscillations in trajectory. The compass constant is set to a high enough value so that the robot very rapidly corrects its position, as evident by no gaps in between error readings. Since the robot corrects itself almost instantaneously, any oscillations in motion are essentially invisible and is therefore critically damped.

Part 3

Question 1

The integral term in PIDs is calculated by adding all prior error measurements. The integral in this case is multiplied by a constant gain, Ki, and there is no need to multiply these discrete error values by a differential. The integral term does not necessarily have to include all prior error measurement. It is possible to reset this integral term every specific time interval in order to avoid extremely large integral values which could potentially skew motion over large distances. On the other hand, the derivative term is found by determining the change in error between time steps and diving by the appropriate time step. For both of these values, the dT value should be used as it represents the time interval between changes in control.

Question 3

The approach I took for tuning the PID parameters followed a similar algorithm as the Ziegler-Nichols method. [3] The approach consisted of the following primary steps:

1. The derivative and integral gains were initially set to 0

2. The proportional gain, Kp, was increased from zero until it reached the threshold between oscillatory behaviour and diverging oscillation. Finding this threshold value was difficult at times due to the noise functions; since they vary between trials and calculations, a specific value of Kp could display oscillatory motion during one trial, and immediately display diverging oscillation if tested again. *Note: The N value was increased in order to recognize oscillations

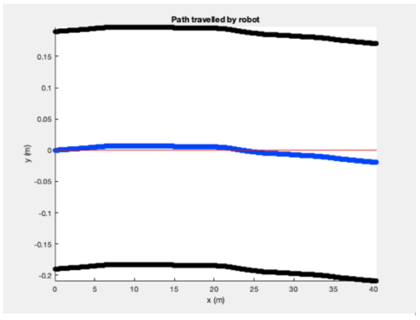


Figure 3- Oscillatory motion at Kp = 195

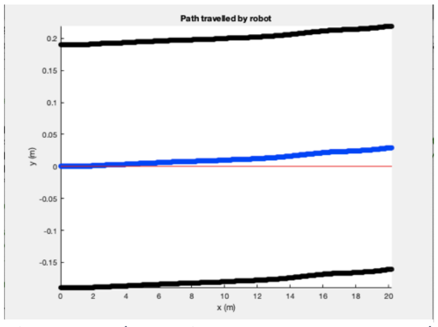


Figure 4- Robot trajectory at Kp = 175.5 and Ki = 0.702

3. This value of Kp is equal to 0.5 Ku, which is defined as the ultimate gain. Tu, defined as the oscillation period was determined from the graph to be roughly 300 seconds in length. Using these values, the Kp and Ki values were determined for PI control. Kp = 0.45Ku and Ki = 0.54Ku/Tu which yielded values of Kp=175.5 and Ki=0.702. This yielded a robot trajectory displayed in figure 4. The purpose of using the integral control is to serve as a method for dampening the oscillation of motion with only proportional control. In recognizing the difference between figure 4 and 5, the robot no longer moves back and forth along the desired path and rather moves in a straight line, although not completely along the desired path. This behaviour is due to the introduction of integral control which further dampens the system, and moves from an underdamped to a more critically damped system.
4. Finally, the value of Kd was calculated in order to establish full PID control. Kp = 0.6Ku, Ki = 1.2Ku/Tu, and Kd = 3KuTu/40 which yielded results of Kp = 234, Ki = 1.56, and Kd = 8775. However, using this extremely high value of Kd lead to very inconsistent graphs. In the majority of PIDs, the oscillations tend to be relatively quick, which did not occur in this case, leading to a high Tu value resulting in a high Kd value. Oscillations in this particular system were not rapid because error was calculated using encoder values rather than deviation from the initial path (this is explored further in question 5). As such, the value of Kd was determined using a trial and error approach with the calculated values of Kp and Ki.

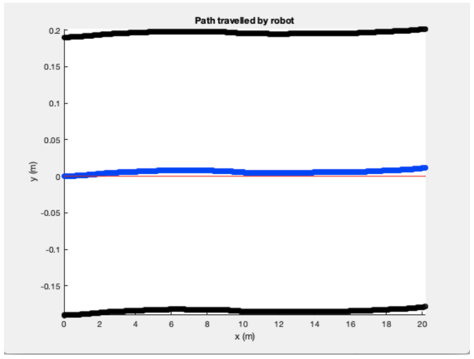


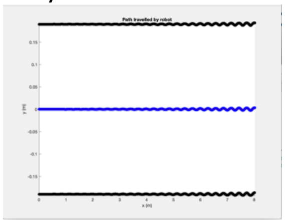
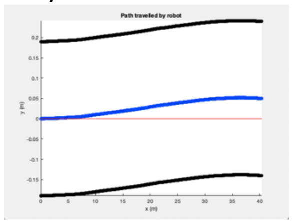
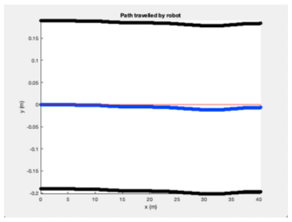
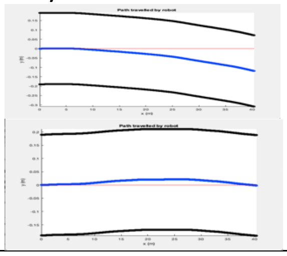
Figure 5- Robot trajectory at Kp = 234, Ki = 1.56, and Kd = 0.01

5. The final gain values used were Kp = 234, Ki = 1.56, and Kd = 0.01. In my opinion, the value of Kd was significantly smaller than the others due to the error and motor power inconsistencies. Since the motors are repositioned during such a small interval of time, it does not require a significant change in error (surface, encoder, or motor power inconsistency) to yield a high slope of the error function. As such, the value of Kd was kept to a minimum in order to enhance the overall trajectory of the robot (figure 5). *Ziegler-Nichols was also completed using compass (Appendix 1)*

Recognizing underdamped, overdamped, and critically damped systems is quite difficult for this particular PID control because the robot initially begins on its desired path and error is calculated using encoder values. Oscillatory motion is usually more visible when the starting position of the system is a considerable distance from the desired values. This is because too high of a Kp value causes an overshoot of the desired position and hence leads to oscillations (underdamped). Similarly, if the Kp value is too low, the system takes too long to respond to the desired position (overdamped). [4]

The threshold value in step 2 is the value that separates critical damping from underdamping. When Kp is set to higher than 195, the proportionality constant so high that it causes the system to hardly veer from the desired path when the initial position is in line with the desired position. However, if the robot were to start a certain offset away from the desired path, too high of a proportionality gain will cause the robot to spin continuously and never return to its original position (hence overdamped, similar to a harmonic oscillator). The addition of the integral term accelerates the response to the desired path by eliminating the residual error of the system, thus further dampening the system. Finally, the use of the derivative term is used to predict future error, which can be seen in the difference between figure 5 and 6. The derivative term is often unstable and used in association with low-pass filters to counter this behaviour. This is the reason the system already works fairly well with simply PI control for critical damping. [5] *Note: error dynamics of tuned PID in Appendix 3

Question 4

Only P Control	Only I Control	PI Control	Only D Control
			
Only using proportional control can lead to rapid oscillations if the robot veers from its desired path	Only using integral control can work in this scenario since the robot begins on its desired path, however, if the robot deviates from the path, the correction time is prolonged	Most systems can work perfectly fine with simply PI control as displayed, however, utilizing D can provide additional dampening control to counteract motion away from equilibrium	Using only D control can provide very inconsistent results. The D gain acts very quickly which can cause substantial issues with short-term changes

Question 5

The disturbances and noise in the current settings already create inconsistencies which cause deviation from trial to trial. One simply manipulation is to increase the rate of noise in the my_noise.m function. Figures 6 and 7 demonstrate the effects of noise on the simulation.

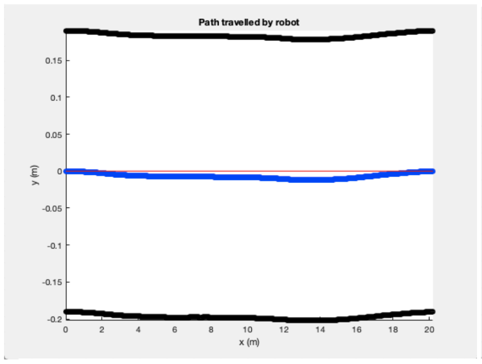


Figure 6- sqrt(0.01) noise constant

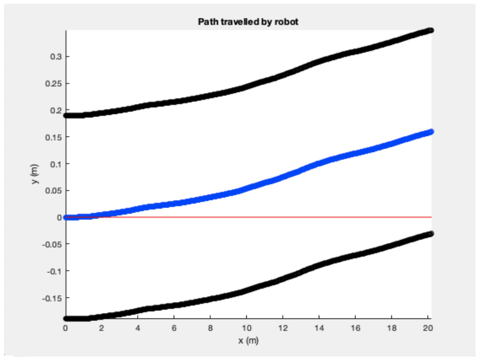


Figure 7- sqrt(0.05) noise constant

As shown, even slightly increasing the level of noise can yield very inconsistent results. Even the initial noise conditions are capable of creating undesirable trajectories through the use of encoders. The use of encoders in general has limitations including changes in path and potential for wheel slippage providing data discrepancies. The PID model was enhanced through the use of a compass, which was able to localize the robot's trajectory as a displacement angle from its original heading. Similar to PID control, the trajectory of the robot was compared to a reference and its position was adjusted accordingly to yield far superior results over longer travelling distances as well, as shown in figure 8.

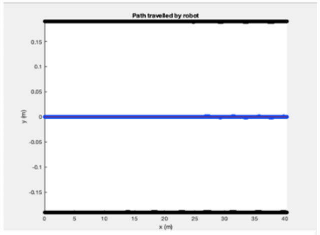


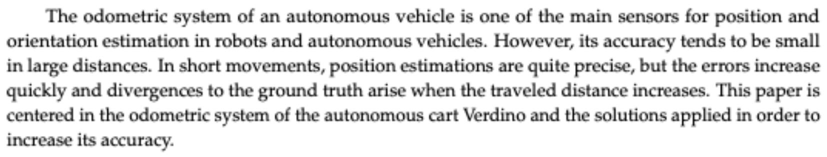
Figure 8- compass control

Question 6

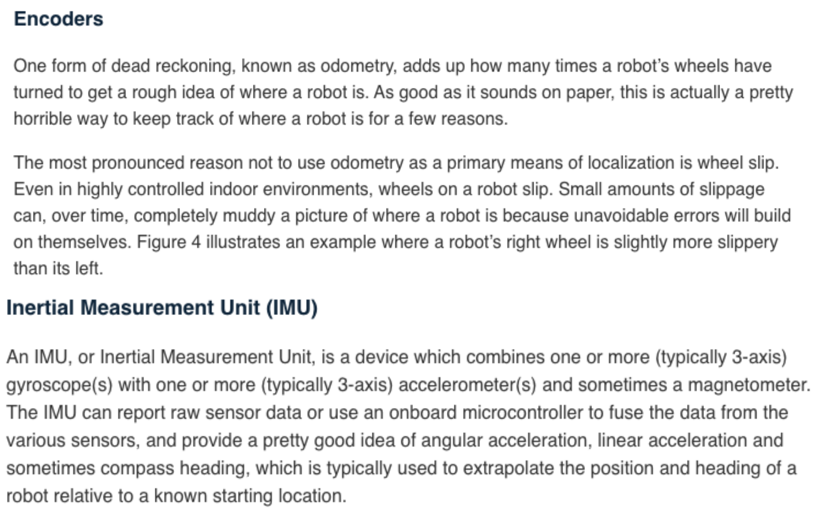
Windup is a practical issue that could arise in the implementation of PID. If the error accumulates over a substantial period of time, its magnitude in conjunction with PD control can cause the system to saturate to some particular physical limit. [6] Derivative noise is also an issue to keep in mind during real world implementation. As was mentioned earlier, utilizing a filtering algorithm can help minimize these fluctuations and allow the system to operate as expected. [7] (view newPos.m)

References

[1] J. Toledo, J. D. Pineiro, R. Arnay, D. Acosta, and L. Acosta, “Improving Odometric Accuracy for an Autonomous Electric Cart,” Sensors, vol. 18, no. 2, p. 200, Dec. 2018.


The odometric system of an autonomous vehicle is one of the main sensors for position and orientation estimation in robots and autonomous vehicles. However, its accuracy tends to be small in large distances. In short movements, position estimations are quite precise, but the errors increase quickly and divergences to the ground truth arise when the traveled distance increases. This paper is centered in the odometric system of the autonomous cart Verdino and the solutions applied in order to increase its accuracy.

[2] S. Krause, “Choosing the Best Sensors for a Mobile Robot, Part Two,” FierceElectronics, 29-Sep-2017. [Online]. Available: https://www.fierceelectronics.com/components/choosing-best-sensors-for-a-mobile-robot-part-two. [Accessed: 03-Apr-2020].


Encoders

One form of dead reckoning, known as odometry, adds up how many times a robot’s wheels have turned to get a rough idea of where a robot is. As good as it sounds on paper, this is actually a pretty horrible way to keep track of where a robot is for a few reasons.

The most pronounced reason not to use odometry as a primary means of localization is wheel slip. Even in highly controlled indoor environments, wheels on a robot slip. Small amounts of slippage can, over time, completely muddy a picture of where a robot is because unavoidable errors will build on themselves. Figure 4 illustrates an example where a robot’s right wheel is slightly more slippery than its left.

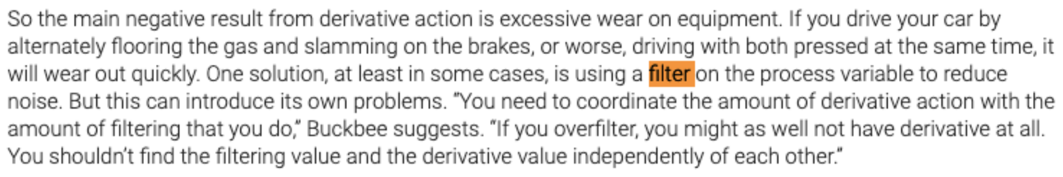
Inertial Measurement Unit (IMU)

An IMU, or Inertial Measurement Unit, is a device which combines one or more (typically 3-axis) gyroscope(s) with one or more (typically 3-axis) accelerometer(s) and sometimes a magnetometer. The IMU can report raw sensor data or use an onboard microcontroller to fuse the data from the various sensors, and provide a pretty good idea of angular acceleration, linear acceleration and sometimes compass heading, which is typically used to extrapolate the position and heading of a robot relative to a known starting location.

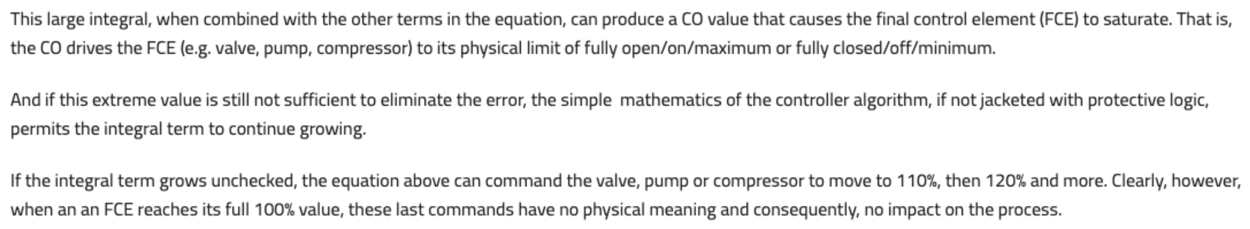
[3] “Ziegler–Nichols method,” Wikipedia, 01-Feb-2020. [Online]. Available: https://en.wikipedia.org/wiki/Ziegler–Nichols_method. [Accessed: 03-Apr-2020].

[4] “Controlling Self Driving Cars,” youtube, 21-Jul-2015. [Online]. Available: https://www.youtube.com/watch?v=4Y7zG48uHRo&t=93s. [Accessed: 03-Apr-2020].

[5] “Understanding Derivative in PID Control,” Control Engineering, 01-Feb-2010. [Online]. Available: https://www.controleng.com/articles/understanding-derivative-in-pid-control/. [Accessed: 03-Apr-2020].


So the main negative result from derivative action is excessive wear on equipment. If you drive your car by alternately flooring the gas and slamming on the brakes, or worse, driving with both pressed at the same time, it will wear out quickly. One solution, at least in some cases, is using a filter on the process variable to reduce noise. But this can introduce its own problems. "You need to coordinate the amount of derivative action with the amount of filtering that you do," Buckbee suggests. "If you overfilter, you might as well not have derivative at all. You shouldn't find the filtering value and the derivative value independently of each other."

[6] Controlguru, “Integral Action and PI Control,” Control Guru. [Online]. Available: https://controlguru.com/integral-reset-windup-jacketing-logic-and-the-velocity-pi-form/. [Accessed: 04-Apr-2020].

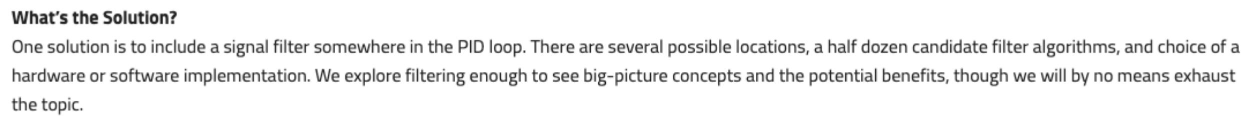


This large integral, when combined with the other terms in the equation, can produce a CO value that causes the final control element (FCE) to saturate. That is, the CO drives the FCE (e.g. valve, pump, compressor) to its physical limit of fully open/on/maximum or fully closed/off/minimum.

And if this extreme value is still not sufficient to eliminate the error, the simple mathematics of the controller algorithm, if not jacketed with protective logic, permits the integral term to continue growing.

If the integral term grows unchecked, the equation above can command the valve, pump or compressor to move to 110%, then 120% and more. Clearly, however, when an an FCE reaches its full 100% value, these last commands have no physical meaning and consequently, no impact on the process.

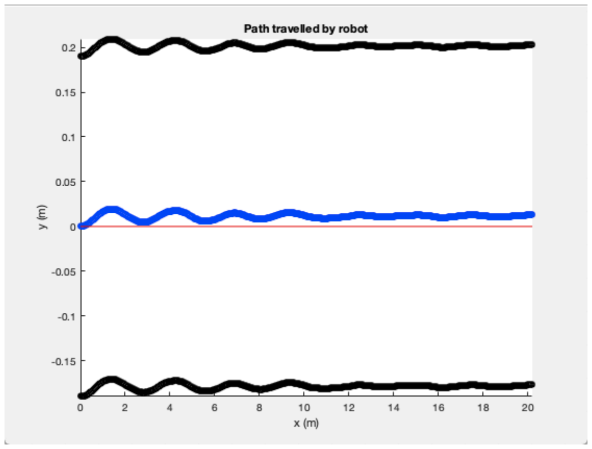
[7] Controlguru, "Derivative Action and PID Control," Control Guru. [Online]. Available: https://controlguru.com/measurement-noise-degrades-derivative-action/. [Accessed: 04-Apr-2020].



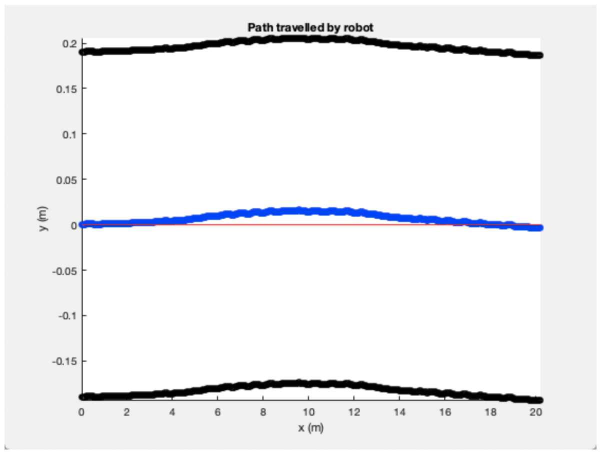
What's the Solution?

One solution is to include a signal filter somewhere in the PID loop. There are several possible locations, a half dozen candidate filter algorithms, and choice of a hardware or software implementation. We explore filtering enough to see big-picture concepts and the potential benefits, though we will by no means exhaust the topic.

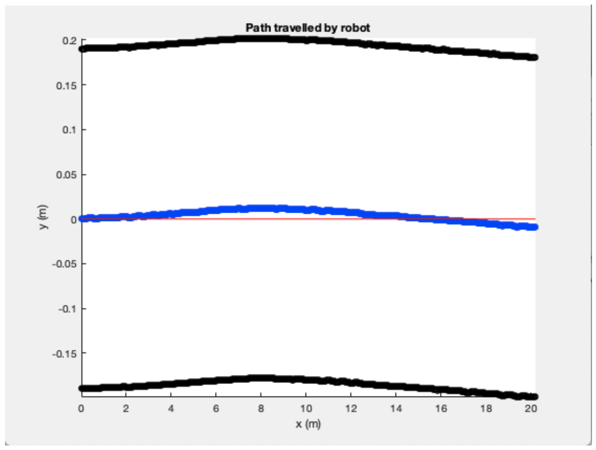
Appendix 1 – Ziegler-Nichols Method with Compass Control



Kc and Kp control



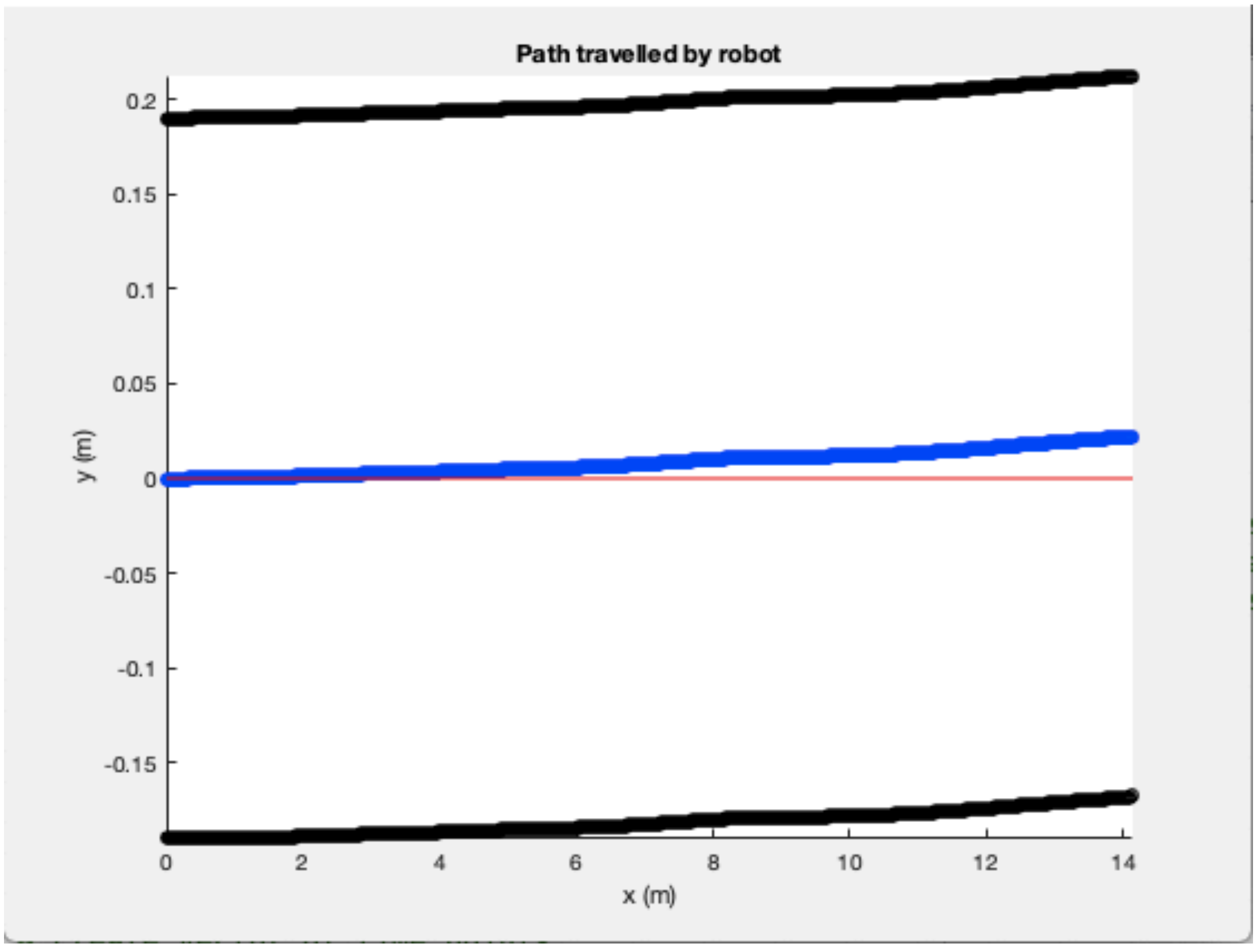
Kc, Kp, and Ki control



Kc, Kp, Ki, and Kc control

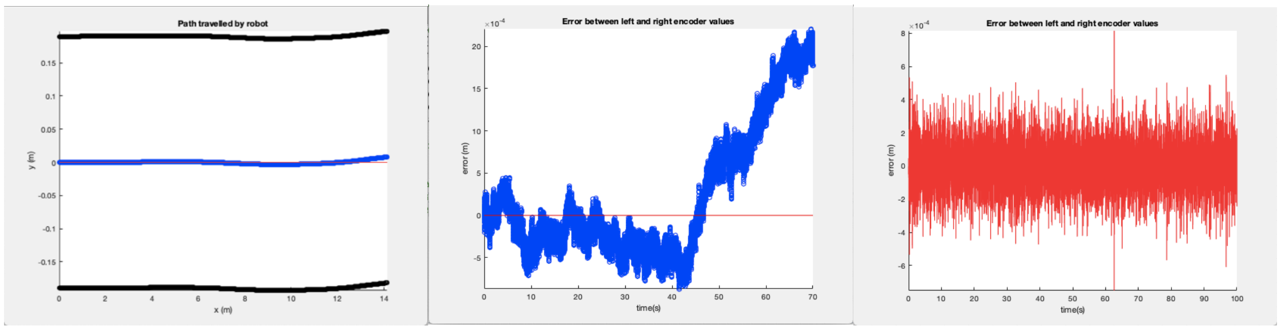
NOTE: THE ABOVE FIGURES WERE OBTAINED BY FOLLOWING THE SAME ALGORITHM IN SECTION 3.3. THIS WAS DONE IN ORDER TO OBTAIN OSCILLATORY BEHAVIOUR WHICH MAKES THE GAIN CONSTANTS EASIER TO DETERMINE

Appendix 2 – PID Control with Filter



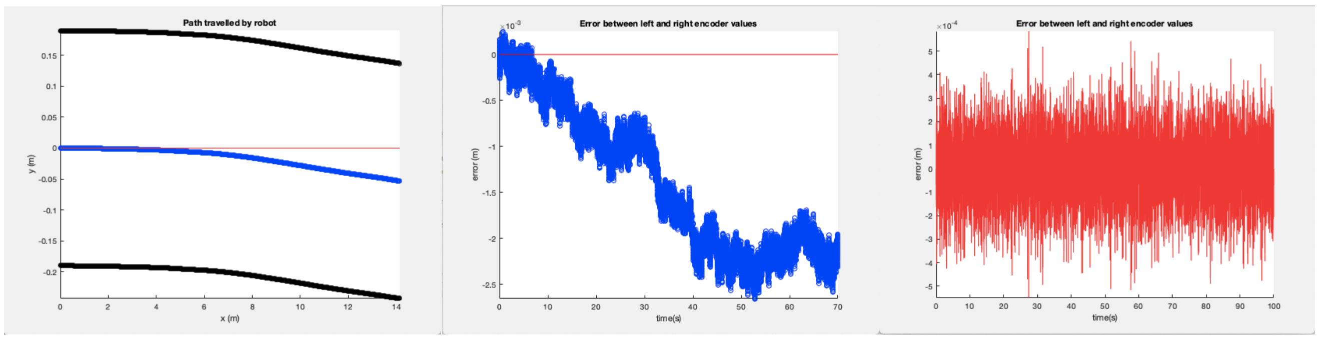
NOTE: THE ABOVE SIMULATION WAS COMPLETED USING THE SAME PARAMETERS AS PART 3 QUESTION 3 WHILE USING AN IMPLEMENTED FILTER (TIME OF SIMULATION: ~1.5 MINUTES)

Appendix 3 – Error Dynamics of Tuned PID



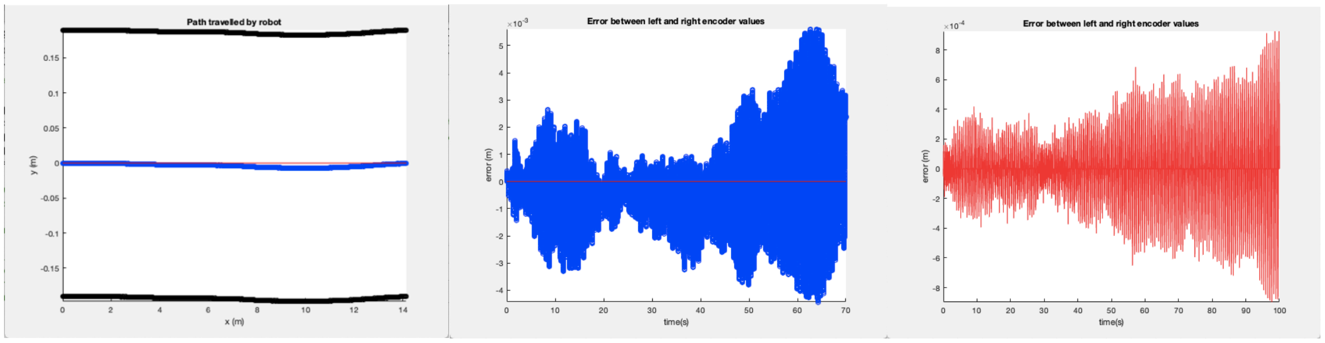
ERROR DYNAMIC OF PID CONTROL, FEW OBSERVATIONS:

- Scale of error between encoders is significantly less than with only proportional control (section 3.1)
- Systems is much more responsive to slight changes in error → leading to a lesser degree of deviation from the desired path
- Enc_error versus is time is much more concentrated around 0, signalling greater control of the robot around the desired path



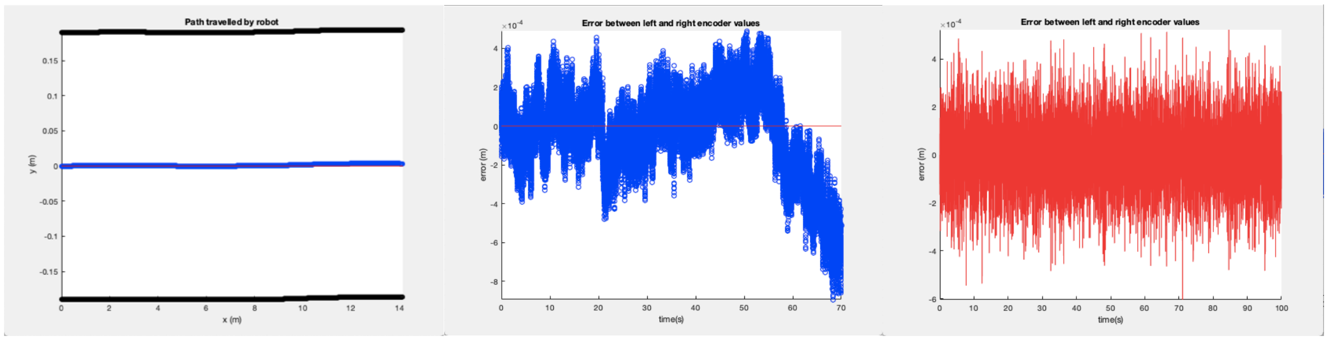
ERROR DYNAMIC OF PD CONTROL, FEW OBSERVATIONS:

- Scale of error is larger in comparison to full PID control as shown above
- Missing integral control is apparent → excessive build-up of error in favour of the right encoder is not taken into account → robot takes longer to correct its position
- Relatively similar enc_error versus time graph

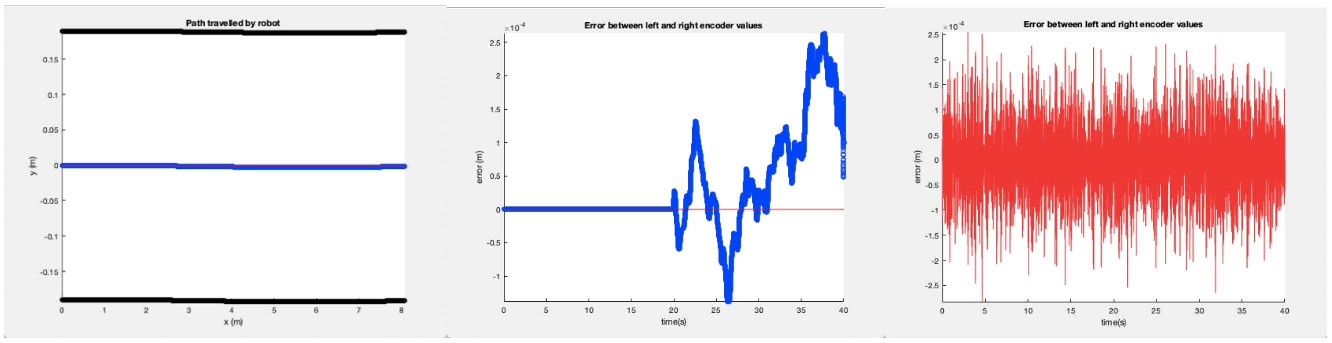


ERROR DYNAMIC OF I CONTROL, FEW OBSERVATIONS:

- Scale of error essentially the same as PD control
- System does a decent job of travelling straight, however, there is a visibly long period between the system recognizing the buildup of error on the right encoder to when it begins correcting its position
- Enc_error versus time displays the exact nature of I control: cumulative error over time (oscillations growing larger) and slow response to deviation (obvious gaps between error readings)



ERROR DYNAMIC OF PID CONTROL WITH COMPASS: NEARLY PERFECTLY STRAIGHT MOVEMENT



ERROR DYNAMIC OF PID CONTROL WITH FILTER: EFFECTS OF NOISE SIGNIFICNTLY DIMINISH AS SHOWN BY ERROR (SMALLER SCALE FOR ENC_ERROR VERSUS TIME)