Closed-Loop Control

Contents

1 Part I: Control Strategies

Before attempting to control the robot, it is first necessary to understand the behaviour of its actuators and sensors.

1.1 Actuators

The example program - `com2009-3009_ev3dev_test.py` - shows how the robot can be driven in a straight line by sending the same value to each motor. However, you need to be able to steer the robot by controlling the wheels 'differentially'. This can be achieved as follows ...

Let v be the base speed of the wheels and δ_v the speed differential between the two wheels, i.e. $v_L = v + \delta_v$ and $v_R = v - \delta_v$, where v_L and v_R refer to the left and right wheels respectively. If $\delta_v = 0$ and $v \neq 0$, then the robot will travel in a straight line. If $v = 0$ and $\delta_v \neq 0$, then the robot will rotate on the spot. If $v \neq 0$ and $\delta_v \neq 0$, then the robot will move along a curved trajectory.

- ☐ Implement the above scheme on the robot and confirm its correct operation by experimenting with different values for v and δ_v. Note that large values of v_L and v_R will be hard-limited by the maximum speed of the motors.

Question 1: *What happens when $v = \pm\delta_v$?* (Worth up to 2 marks)

> Let the speed of left wheel be $v_L = v + \delta_v$ and the speed of the right wheel be $v_R = v - \delta_v$
>
> When $v = \delta_v$, $v_L = 2\delta_v$ and $v_R = -2\delta_v$, i.e both of these wheels are moving in opposite directions but with equal magnitude of velocities. The translational velocities in opposite directions generates torque on the (imaginary) axle of the robot and causes the robot to rotate about a central axis with maximum angular velocity. This means that the robot is able to change it's direction the quickest at $v = \pm\delta_v$. The sign (+) indicates direction of rotation, a positive δ_v would lead to clockwise rotation and a negative δ_v would lead to anti-clockwise rotation.

1.2 Sensors

The example program - `com2009-3009_ev3dev_test.py` - shows how to read the output of the ultrasonic sensor. However, for real-time control, sensor outputs need to be sampled at regular intervals.

- ☐ Implement an *inner* processing loop that samples the output of the ultrasonic sensor once every 10 msec.
- ☐ Implement an *outer* processing loop that calculates the running[1] mean, standard deviation, minimum and maximum values from the sensor over a period of 10 secs (i.e. 1000 samples). On completion, display the results on the EV3 screen and/or the `VSCode` debug window.
- ☐ Experiment with your robot facing various surfaces.

[1] Here's how to calculate a 'running' mean and standard deviation: `https://www.johndcook.com/blog/standard_deviation/`

Question 2: *How do the mean, standard deviation, minimum and maximum values from the ultrasonic sensor vary as a function of the actual distance to a surface?* (Worth up to 4 marks)

Actual Dist.(cm)	Mean(cm)	Std. Dev(cm)	Min(cm)	Max(cm)
1	255.0	0	255.0	255.0
20	22.6	1.9	22.4	23.3
30	29.3	0.15	29.1	29.9
35	33.97	2.26	32.8	35.3
90	89.5	0.12	89.2	89.6
150	152.1	3.1	151.2	152.9
160	159.7	4.5	159.0	161.0
200	193.5	22.53	176.0	244.3
240	235.7	31.55	182.0	255.0
250	255	0	255.0	255.0

Table 1. Comparison of actual distance vs. measured distance means/standard deviations of 1000 samples and the min/max values measured.

From the results above, the following conclusions can be inferred

- Between actual distances of 20cm to 150cm, the standard deviation is minimal and the differences between the minimum sampled value and the max. sampled value are minimal. With respect to the actual distances to the surfaces, the means and standard deviations remain fairly consistent, and so does the differences between the min. measured value and the max. measured value.
- Between actual distances of 200-250cm, the standard deviations are much larger and the differences between min. sampled value and max. sampled value are comparatively larger. This suggests that for a distance of 200cm, at one instance it's possible to get a reading of 176cm and a reading of 244cm at another instance. Given actual distances, the mean measured distance becomes more inconsistent as the distance increases. This is also true for standard deviation. The difference between the min. value and the max. value also increases with respect to actual distance.
- Anomalous readings for 1cm and 250cm , this is due to limitations of the ultrasonic sensor. It is not possible to read any distance less than 1.9cm as this is shorter than the ultrasound wavelength and the sensor would not be able to receive it, hence a reading of 255cm, thinking that the object is far far away, when in fact, it is close.

The practical results obtained from our robot showed that the closest a sensor was to a surface, the closer the mean was to the actual value, with the standard deviation being lower with the minimum and maximum values not varying as much from the mean. Once the sensor was moved back, the deviation increased, and once the threshold value of 250cm was passed, all values were set at 250cm. Hence the measured values are directly correlated with the distance.

Question 3: *How do the mean, standard deviation, minimum and maximum values from*

the ultrasonic sensor vary as a function of surface angle? (Worth up to 4 marks)

Actual distance from obstacle: 30cm

Angle (degrees)	Mean (cm)	Std. dev (cm)	Min(cm)	Max(cm)
15	28.9	0.095	28.9	29.2
30	32.59	0.39	32.1	33.2
39	32.6	0.52	32.1	36.1
45	255.0	0	255.0	255.0

Actual distance from obstacle: 80cm

Angle (degrees)	Mean (cm)	Std. dev (cm)	Min(cm)	Max(cm)
30	79.44	2.99	78.8	80.6
45	58.1	121.2	42.9	68.9
60	255.0	0.0	255.0	255.0

Table 2 and 3. Comparison of angles vs. measured distance means/standard deviations of 1000 samples and the min/max values measured.

It was observed that as a function of surface angle, the mean distance becomes more inconsistent as the surface angle is increased. At a certain angle, somewhere between 40 degrees and 60 degrees, the sensor is not able to receive any ultrasonic that it emits, therefore failing to work.

- When obstacle was at 30cm , it was observed that although the distances were accurate for angles up to 39 degrees, at 45 degrees it failed to provide a correct reading. As a function of surface angle, the mean distance measured started to become more inconsistent, and standard deviation, and the differences in minimum/maximum values also increased, leading to what can be assumed as an exponential jump from 39 degrees. To confirm our hypothesis, we decided to test it at a different distance
- When the obstacle was at 80cm , the mean distance measured started to become more inconsistent with respect to an increase in surface angle. The same observations from earlier held up for standard deviation and differences in minimum/maximum values. What was interesting was, the angle at which the sensor failed was >45 degrees. This is probably because an average ultrasonic sensor has a beam angle of 10-15 degrees. At angles, the waves reflect and cancel out each other, leading to the case where the sensor never receives any of the reflected beams. We believe this is why the sensor fails at larger angles.

It is very clear that as the surface angle increases, the measurements became increasingly inaccurate, which was worse with larger distances from the surface.

1.3 Open-Loop Control

☐ Remove the ultrasonic sensor from your robot.

☐ Write a program that causes the robot to travel 0.5m in a straight line, rotate 180°, retrace its path, rotate 180° (and so on) continuously.

☐ Experiment with different surfaces.

Note: *Don't spend too long on this. You'll find that it's quite difficult to achieve a satisfactory solution - and that's the point!*

Question 4: *In the above implementation, (i) how did you determine the control parameters for ensuring that the robot travelled the target distance and rotated the appropriate amount, (ii) how sensitive was it to the selected parameters, and (iii) how reliable was the best solution?* (Worth up to 4 marks)

i) We decided it would be best to keep the speed as a constant, and measure how fast the robot travelled in 1, 2, and 3 seconds respectively, as the robot did not seem to consistently move the same distance. We found the robot roughly travelled 0.25m in 3 seconds, hence we set a loop for 6 seconds. To turn the robot 180 degrees, we measured how much it turned in 1 and 3 seconds using the same speed above (20), and found it took 2.3 seconds to complete a 180 degree turn.

ii) The robot was very sensitive to the selected parameters, if the robot were to run for half a second longer, it would turn the robot around 220 degrees and in the case where the robot is going straight, it would make it travel a further 7cm.

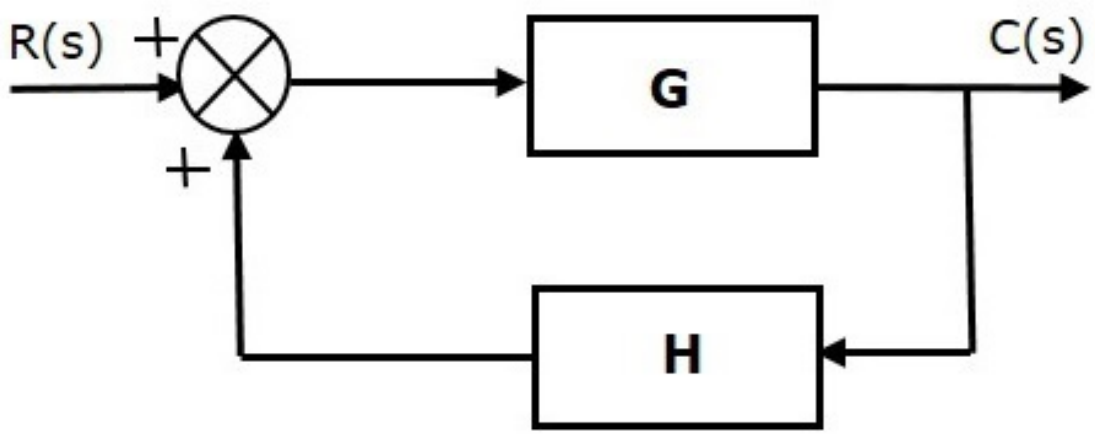
iii) The solution was not very reliable, in fact, the robot behaved differently on different surfaces with the selected parameters, hence limiting our parameters to just one type of surface. However, the solution is easy to code and adaptable to other situations once we had found what time values led to the desired outputs.

1.4 Closed-Loop Control

In the control system you implemented above, you will have discovered that after a number of iterations backwards and forwards, the robot was no longer following its original track. Any disturbances (e.g. due to wheel slippage or variations in surface texture) and/or inaccuracies in the control parameters would cause the robot to deviate from its intended path. This is because the robot had no information about its position or orientation, i.e. it was using 'open-loop' control (otherwise known as 'dead reckoning').

Clearly, if a robot had 'feedback' from the environment, then it could use that information to maintain its intended path, i.e. 'closed loop' control. In particular, closed-loop control actions can be based on minimising the *difference* between a target value and a sensed value, and this is known as 'negative feedback' control.

A classic closed-loop negative-feedback control system is structured as follows:



... where the input reference signal r specifies the 'setpoint' for the system, i.e. the target value of the feedback signal b. Based on the value of the error signal e, the controller g_1 sends signals to the actuators g_2 causing some behaviour c. Sensors h detect the consequences of the behaviour and provide feedback b, which is compared to the reference r. The difference between the feedback signal b and the reference r generate the error signal e, and the loop continues (iteratively minimising $r - b$).

The controller g_1 is commonly configured as follows:

$$u(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de}{dt}(t),$$

... where K_P is the 'proportional' gain, K_I is the 'integral' gain and K_D is the 'derivative' gain. This is known as a 'Proportional-Integral-Derivative' (PID) controller. Note that a simple controller might only use K_P (i.e. $K_I = 0$ and $K_D = 0$).

Question 5: *In a negative-feedback control system, what is the significance of $e = 0$?* (Worth up to 2 marks)

In a negative feedback loop, when an error is set to 0, this could mean two things depending on the context.

The system has no error and has achieved it's set point value. Therefore the output of the feedback loop will remain constant and will not change unless there is some error in the system.

The error is zero due to a sensor malfunction, and the system thinks it has achieved it's set point value. Therefore, the output of the feedback loop will not respond to any changes to the environment due to a lack of error signal, or feedback.

Question 6: *Give an example of a real-world negative-feedback control system and describe its operation.* (Worth up to 8 marks)

An example of a real world negative-feedback control system is the cruise control system of a car. It works by comparing the actual speed of the car with the desired

speed, whose difference makes up the error signal.

In the case where the cruise control system can only actuate using the accelerator, if there is a negative error (current speed is less than target speed), the accelerator pedal is actuated, thus increasing fuel flow to the engine and increasing the speed. Once the error is less than 0, the target speed is at (or above) the set point speed, so no more acceleration is necessary.

The speed sensor then keeps sampling the speed so when there is an error, implying the speed has fallen below the set value, it then actuates the accelerator.

Question 7: *Assuming that* $K_I = 0$ *and* $K_D = 0$*, what would happen to your example system if (i)* $K_P = 0$ *or (ii)* $K_P < 0$*?* (Worth up to 4 marks)

i) When $K_P = 0$, the error signal is multiplied by a gain of 0 causing the error signal to equal zero. Hence the accelerator would never be actuated and the control system would not do any work to achieve the desired set point speed.

ii) If there were no brakes connected to the system, when $K_P < 0$ nothing would happen as the accelerator cannot slow the car down and cannot be actuated in a way to slow the car down. So if a car was going downhill, the car's speed would increase to a dangerous amount and the system would be unable to appropriately respond to it.

However, if there were brakes connected to the control system then ideally, the error would actuate the brakes and cause the car to slow down until the error became zero, after which the brakes would be disengaged. However, if the car slows down due to too much braking, the sign of the error changes and the accelerator would be actuated. Depending on the parameters of the system, this process of oscillating back and forth with minor actuations of the brakes and accelerator will continue as long as the error is non-zero.

1.5 Basic PID Controller

The next step is to program the robot to position itself *autonomously* at a set distance from a vertical surface (such as a wall). In order to do this you will need to implement a closed-loop negative feedback system incorporating a PID controller.

- ☐ Re-attach the ultrasonic sensor.
- ☐ Implement a PID controller of the form $u(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de}{dt}(t)$, where the 'reference input' r corresponds to the desired distance between the robot and the surface, the 'feedback signal' b corresponds to the output of the ultrasonic distance sensor, and the 'control signal' u corresponds to the value that is sent to the motors. You will have to measure dt (the loop time) using the system clock.

Once you have implemented the code, you should follow these steps:

- ☐ Set $r = 50$ cm.
- ☐ Set $K_P = 1$, $K_I = 0$ and $K_D = 0$.

- ☐ Place the robot 100 cm from a vertical surface with the ultrasonic sensor facing towards it.
- ☐ Run the code.

If all is well, then your robot should move slowly towards the surface slowing down and eventually stopping as it approaches the 50 cm point.

Note: *If your robot appears to do the opposite of what you expect, try setting* $K_P = -1$*. If that solves the problem, then you have implemented a positive-feedback rather than a negative-feedback loop. This means that you need to swap the polarity of the signal being sent to the motors (and reset* $K_P = 1$*).*

Question 8: *What happens if, after the robot has arrived at the 50 cm point, you manually push it towards the surface?* (Worth up to 2 marks)

It resisted our push, increasing it's resistance as we pushed harder, to stay at the 50cm mark, as the error becomes larger the farther it was pushed from the set point. By putting another obstacle in front of the robot, the robot would change it's position and maintain a 50cm distance from the new obstacle.

Now implement an outer loop that swaps r between 30 cm and 50 cm every 10 seconds. As a result, your robot should now change its position at regular intervals. These step changes in r will make it easier to observe the robot's behaviour for different values of K_P, K_I and K_D.

Question 9: *With* $K_I = 0$ *and* $K_D = 0$*, what happens when you experiment with different values of* K_P*? In particular, (i) what is a 'good' value for* K_P *(and why), (ii) what value of* K_P *causes the robot to start to oscillate continuously backwards and forwards around the target* r*, and (iii) when it starts to oscillate continuously, what is the oscillation period? Hint: start with* $K_P = 1$*, then increase it gradually.* (Worth up to 5 marks)

i) We found the 'good' value for Kp using our solutions code to be 0.5, which resulted in no or very minimal oscillations before coming to a standstill. The reason why this is a good value is that there was low or no overshooting at all. The oscillation was nearly zero when reaching the set point distance. The robot would come to a smooth stop when reaching the set point, without any overshoot or quick jittery movements at all, as one would expect from a human driver.

ii) At about a K_P value of 2.5, the system found it difficult to stop at the 50cm target (the robot came to a complete stop only once out of five experiments). At this value, the robot oscillated perpetually back and forth between the 50cm mark and never came to a stop, even after 5 minutes.

iii) For K_P=2.5, we found the oscillation period was roughly 1 second, with some margin of error due to our response times and difficulty in measuring the small oscillations with a mobile timer.

The value of K_P that causes the robot to oscillate backwards and forwards continuously is known as the 'ultimate gain' K_U, and the oscillation period is designated as T_U.

Note: *Be aware that* T_U *is a measure of time, not frequency. I.e. if the robot oscillates*

backwards and forwards at a rate of five times a second, then $T_U = 1/5 = 0.2$ *secs.*

Question 10: *What is the relationship between the 'good' value of* K_P *that you discovered by experimentation and the value of* K_U *that you measured?* (Worth up to 5 marks)

From our measurements we found the good value of K_P to be 0.5 whilst the value of K_U to be 2.5, hence the relationship between the two is $K_P = 0.2\ K_U$, at which there is no overshoot.

The time taken was found to be 1 second for oscillation at K_U, therefore T_U is 1 second.

1.6 PID Tuning

There are several methods available for tuning the parameters of a PID controller. The *manual* method involves setting $K_P = 0.5K_U$, $K_I = 0$ and $K_D = 0$. Next, K_I is gradually increased to improve the convergence. Then K_D is gradually increased to improve the responsiveness.

A more formal tuning approach is known as the *Ziegler-Nichols* method. Once K_U and T_U have been determined, the PID gains may be set according to the following heuristic:

	K_P	K_I	K_D
P	$0.5K_U$		
PI	$0.45K_U$	$0.54K_U/T_U$	
PID	$0.6K_U$	$1.2K_U/T_U$	$3K_UT_U/40$

Question 11: *What values of* K_P*,* K_I *and* K_D *are given by the Ziegler-Nichols method for a P, PI and PID controller (based on the values of* K_U *and* T_U *you measured previously)?* (Worth up to 5 marks)

The value of K_U previously measured is 2.5 and the value of T_U previously measured was found to be 1 second. Using these values.

For a P controller

$K_P = 0.5\ K_U = 0.5\ (2.5) = 1.25$

For a PI Controller

$K_P = 0.45K_U = 0.45\ (2.5) = 1.125$

$K_I = 0.54\ K_U/T_U = 1.35$

For a PID Controller

$K_P = 0.6K_U = 0.6\ (2.5) = 1.5$

$K_I = 1.2K_U/T_U = 1.2\ (2.5)\ /\ (1) = 3$

$K_D = 3K_UT_U\ /\ 40 = 3\ (2.5)\ (1)\ /\ (40) = 0.1875$

☐ Confirm the effectiveness of the Ziegler-Nichols method by setting the values for K_P, K_I and K_D in your controller.

Question 12: *Using the values for K_P, K_I and K_D given by the Ziegler-Nichols method, what happens when you decrease the sampling rate (e.g. by changing the value of the 'wait' function in the loop)? What is the significance of your observations?* (Worth up to 5 marks)

Sampling Rate (Hz)	Time period Avg. (s)	Wavelength Avg. (cm)	Oscillation?
1	4.1	6	Yes
2	3.5	4.4	Yes
3	3.2	4	Yes
4	3.2	4	Yes
50	1.5	Less than 2	Yes
100	1	Less than 1	No after 44s
1000	Less than 1	Less than 1	No after 32s

Table 3. Comparison of sampling rates and average time period/wavelength of oscillations

We started with a sampling rate of 1 Hz (1.0s). We ran the robot for 1 minute total with any given sampling rate. It was found that at 1Hz up to 4Hz the robot was sluggish to notice the changes in the environment, as there was a large delay in the information capture, which caused the robot to move way past the set point. This process would continue and the robot would never stop, oscillating between the set point distance. We increased this to 50Hz and it became marginally better.

As we increased it to 50Hz, the oscillation time period reduced and the length of the wave of oscillation was shorter. As we increased it to 100Hz and 1000Hz, the robot became more responsive to the changes in the environment. However, we noticed that the integral error started to become larger, as a result we had to put in some code to avoid a scenario known as integral windup.

From the above, it is very clear that if one decreases the rate at which samples are taken, the robot becomes less responsive to changes in the environment, due to the delays in taking samples. The oscillations become more amplified and the system becomes more unstable.

The significance is that, as the sampling rate increases, the oscillations are dampened, decreasing the length and time of the oscillations and the robot becomes more responsive and stable to the changes in the environment. However, if the sampling rate is increased beyond a certain point, there is a large risk of integral windup as the errors are added up at a quicker rate.

1.7 PID-based Steering

Finally (for Part I), reposition your ultrasonic sensor so that it faces sideways from the robot.

Now re-program your robot with a PID controller that maintains a fixed distance from a wall as the robot travels along parallel to it. This will require the speed to be set at some fixed value, and the PID-based control loop will handle the steering. Use the Ziegler-Nichols method to determine appropriate values for K_P, K_I and K_D (this can be done while the robot is stationary and changing the target distance a few centimetres

once every 10 seconds, as before).

Note: *It is particularly important that the sensor is mounted well forward of the driving wheels, otherwise it will not be sensitive to changes in direction when the robot is stationary.*

Question 13: *What values for K_U and T_U did you measure, and what are the resulting values for K_P, K_I and K_D given by the Ziegler-Nichols method?* (Worth up to 5 marks)

The measured value for K_U was 2 and the measured value for T_U was 3 seconds

The resulting K_P, K_I and K_D were

$K_P = 0.6\ (2) = 1.2$

$K_I = 0.54\ (2)\ /\ (3) = 0.36$

$K_D = 3\ (2)\ (3)\ /\ 40 = 0.45$

☐ Confirm the effectiveness of these values for K_P, K_I and K_D by having the robot travel along a wall maintaining a constant distance from it.

1.8 Summary

You have successfully completed Part I of the assignment, and you should now have all of the skills necessary to move on to Part II.

2 Part II: Maze-Running Competition

2.1 Objective

The aim of Part II of the assignment is to use the theoretical knowledge and practical experience you have acquired in Part I to design and implement a robot that is capable of competing in a maze-running competition.

The challenge is to navigate a corridor (that will be set up in the robot arena) in the fastest time possible and without touching the walls. The precise layout will not be revealed until the final lab session. However, you will be able to practice in the arena beforehand.

You need to decide which control principle(s) to use and how to set up your robot's sensors and actuators. Be sure to take into account information you have learnt in Part I. Also, since it's a competition, you will need to to think about how your robot manages 'risk', i.e. is it better to be slow-and-careful or reckless-but-fast? You will probably need to experiment with various alternatives before deciding on a final approach.

Note: *You will need to adopt good working practices for (i) organising your team and (ii) software version control. The latter is important as robots are notorious for failing after a so-called 'improvement', so being able to revert easily to an earlier version will save a lot of pain and grief.*

Note: *Each Lego kit contains two ultrasonic sensors.*

Note: *The maze will be a long winding corridor. There will be no dead-ends, no loops and no junctions. However, there may be chicanes and small breaks in the wall.*

2.2 The Competition

In order to give every team an equal chance, the competition will be organised into a number of leagues, each consisting of several teams (and their robots). Marks will be awarded based on each robot's performance **within its league**. This will mean that each team will be competing against teams that have had an equal amount of time devoted to the development of their respective robots. The competition will take place during the final lab session.

Note: *If there is time at the end of the league battles, the winning robot in each league will compete again to find the overall champion. This 'championship' contest will not count towards the marks for the assignment, but a (small) prize may be awarded.*

As you can imagine, running such an event with a large number of teams/robots requires very precise time management, so we will be issuing a strict timetable for the final lab session. This should appear on MOLE the week before. It is essential that you prepare carefully for your designated time slot (otherwise you may lose marks - see below).

The rules for the competition are as follows:

1. Any number of practice runs may be made (subject to the availability of the arena).
2. Each team must register their arrival at the arena (with their robot) 10 mins prior to their league's designated time slot, after which no further technical development will be permitted.
3. Each team will be called forward in turn to place their robot on the starting line.

4. An 'official' run for each team's robot will be timed by the lab demonstrators.
5. If a robot fails to complete a run within a **2 minute time limit**, the demonstrators will record the furthest position it reached.
6. Each 'wall touch' will incur a **10 sec penalty**.
7. If a robot needs to be 'rescued', a *designated* team member may place it back on the course at the location where things went wrong. However, the clock will keep running.
8. The designated rescuer must stand *outside* the arena next to the starting position, and return to that position after each rescue.
9. Up to two rescues are permitted.
10. Each rescue will incur a **20 sec penalty**.
11. A third rescue is NOT permitted. Instead the run will be terminated and the position reached will be recorded.
12. Teams will be ranked in their league according to their finishing time (including any penalties) or their distance travelled.
13. Teams who travelled the same distance will be ranked according to the time taken to get to that point (including any penalties).
14. A team which fails to arrive at the arena at the designated time will incur a **60 sec penalty** and may NOT get a run.

Marks will be awarded as follows:

- 15 marks will be awarded to the team with the best run within their league, 9 marks to the second best team, and so on.
- 5 *bonus* marks will be awarded for a run completed within the 2 minute time limit with *no* wall touches and *no* rescues.
- A team who fails to appear at the starting position will receive 0 marks.

2.3 Your Team's Solution

Question 14: *In terms of your robot's software architecture, what principles did you explore, and what was the final approach taken?* (Worth up to 15 marks)

Initially, we wanted to use just one sensor using the single wall follower's PID values, as this was working without issues. However, we decided against this as it was obvious that the robot would lock on to the wall behind the one it lost on a sharp turn, making the robot do a 180 and lock on to the opposite wall it had come from, going back on the path it had just followed.

Once we added two sensors we tried to use the same wall follower code from before, but using the left sensor's error to turn right and using the right sensor's error to turn left. When we implemented this, we saw that the robot had very high oscillations, and these two separate control loops were fighting over each other to maintain a fixed distance on both sides, which was simply not possible, as when the robot would get

close to one of the walls, the error for the other wall would be too large, forcing the robot to go to the other wall. This system was largely unstable.

The control loops "fought" for control of the wheels, alternating between skewing the path left and right, leading to massive oscillations and sometimes crashing into the wall. The prime issue with this algorithm was, as the sensors got too close to the wall, there would be a very sharp overshoot to the said wall, as the ultrasonic sensor would think that the obstacle is very far away, when in fact the obstacle is very close but the ultrasonic sensor is unable to detect it due to the flaw discussed in Question 2.

At this point we realized we needed a different way of measuring the error. We tried to find the "midpoint" of the maze by summing the obstacle distanced received by the left ultrasonic sensor and the right ultrasonic sensor using the following metric.

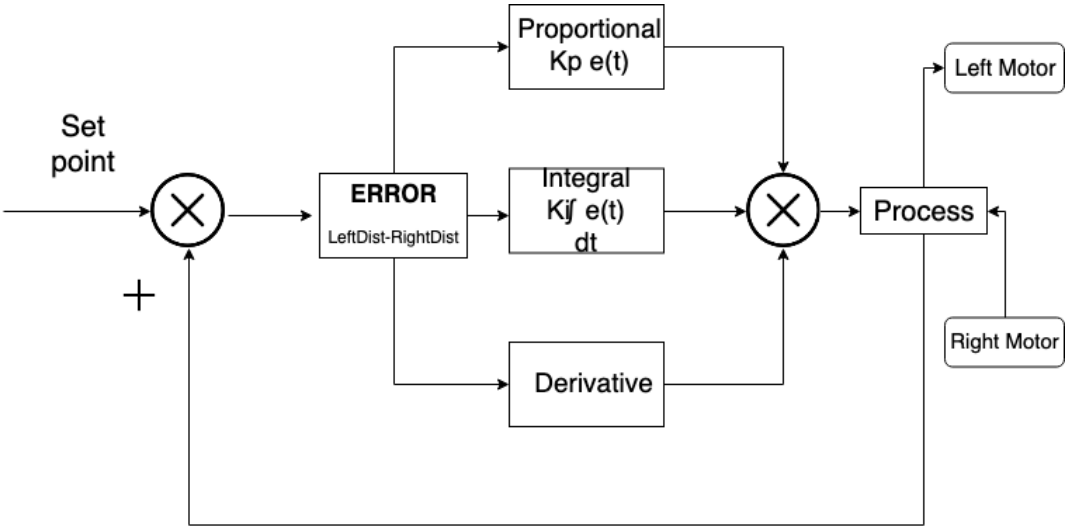
$$Error = \frac{LeftSensorDist + RightSensorDist}{2}$$

We used this error as the minimum distance to keep from the obstacles. The main issue with this metric was if one of the walls were too far, the midpoint distance would be very large, and the robot would just crash into the closest wall, which was very undesirable. This metric did not model the midpoint correctly.

After some rigorous trial and error, we figured that the robot would be at a midpoint between the two walls if and only if the differences between the distances between the left wall and the right wall measured from the midpoint were zero. We decided to use this as our error metric and we found out that the following metric models the midpoint correctly

$$Error = LeftSensorDist - RightSensorDist$$

In this metric, if the error was negative (leftside wall is closer than rightside wall), it hints that the robot was too close to the left and would bias the left and right motor speed offsets to turn right and make the error zero. Similarly, if the error was positive, it would bias the left and right motor speed offsets to turn left. If both obstacles were at the same distance, the error would be zero and the robot would follow the ideal midpoint path. Simple, right? If one of the sensors provided an erroneous value when the sensor is very close to the wall (255cm), we made a function to replace this value with the correct value by comparing previous readings to figure out if the robot was approaching or moving away from an obstacle. Finally, after some PID tuning, this metric gave the best performance in the maze.



PID Block Diagram for Robot

For our final robot architecture, we decided to implement an edited level 0 control system, as we felt it was more then suitable to out task due to the robot not having to search the maze but follow a linear path to the end.

In the diagram above,

- The ultrasonic module takes a distance reading of the left and right side of the robot
- The difference metric uses the final error metric discussed previously.
- The Time module measures the time from the previous sensor measurement down to the last ms.
- The integral module uses the formula mentioned above to derive a value for the dv to be applied to each motor.
- The actuate module applies the dv to the default speed in each motor (using -1*(v-dv) for the left motor and -1*(v+dv) for the right motor) and uses this value as the speed to be applied to each robot wheel.

We did not implement any "wanderer" modules because as long as our robot consistently moves forward in this "maze" it will never have to worry about changing it's direction. If the maze was a bit harder, we could have potentially used an odometry approach to remember whenever a choice of direction had been made, but since for our task it was a linear path, we didn't feel the need to complicate our solution.

Question 15: *What was your robot's final physical configuration (include a photo)?* (Worth up to 5 marks)

The robot consisted of two sensors mounted close to the robot on the front side, at an angle no more than 40 degrees, so that it can see ahead of a curve and make adjustments to the motors accordingly. It was connected to two normal wheels directly to a two large motors. No gearings were used. To improve stability for the ultrasonic sensors, we made a rigid frame so that the sensors would stay intact when the robot

is moving and turning at high speeds. The sensors were kept as close to the robot as possible, so that the distance measured from the wheelbase would be as accurate as possible, otherwise we would have to consider the distance from the wheelbase and the sensor and would have to subtract it from the readings.

We have the sensors parallel from the robot, so that the distance measured is not in front of the robot but on the sides. The sensors are angled towards the front at an angle no more than 40 degrees, so that it is possible to look ahead for the curvature of the walls and turn before the robot has entered into a curve;

Question 16: *Are there any points you wish to make about your robot's performance in the competition? For example, if you had had more time, what might you have done differently?* (Worth up to 5 marks)

The robot's performance in the competition was exceptional, being able to navigate its way through the maze in 32 seconds with no intervention nor touching any of the sides. We came 2nd in the final competition at 46 seconds. Our robot followed the middle of the path quite well, not coming dangerously close to walls when the passage became very narrow.

Had we had more time, the overall speed of the robot could have been increased along with potentially tuning the controller in order to find a faster solution. We would have also considered using gears to get past the 100 RPM limit of the motors to increase the speed of the wheels. In short, if we had more time - we could have figured out new PID parameters for a faster maze runner.

If we had lots of time, it would be possible to look into neural networks instead of using PID to solve this navigation problem by mining the data measured by the two ultrasonic sensors and doing simulations in a virtual world, similar to what self-driving companies like Google-owned Waymo are doing to train their vehicles.

Question 17: *What was the result of your official attempt?* (Worth up to 20 marks)

Distance	**Time**	**Wall Touches**	**Rescues**
Finished	00:32	None	None

Note: *The demonstrators will have already recorded the above information. However, please include it here as a cross-check.*