Design Principles and Methods

ECSE211

Tony Ou

260867785

Hong Tao Xu

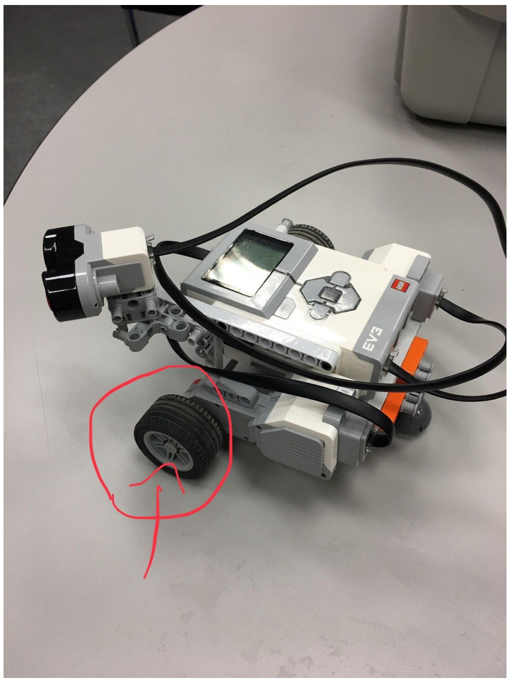
Lab 1 – Wall Following

Group 27

Design Evaluation

Hardware Design

We started the hardware component of the lab by trying out different prototypes inspired from the internet. We then discussed the advantages/disadvantages of each and settled on the best one, which had the most stable placement of the EV3 brick and regulated motors. After implementing the right design, we had to test out different sensor placements so that it stays still at the right height and at the right angle to get the most accurate readings. We then ran multiple test runs using the blocks used in the demo. By doing so, we discovered flaws on the robot, such as the stick holding the wheels being too long and the sensor moving during turns. These issues were fixed by adjusting the length of the stick and using pegs to block the rotation of the sensor, as shown in Fig. 1.1 and Fig. 1.2 respectively.



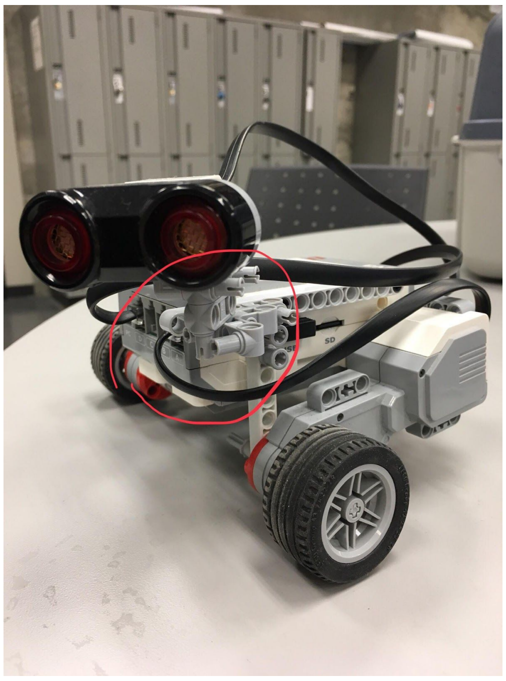


Fig. 1.1 Wheel stick adjustment

Figure 1.2 Pegs for the sensor

Software Design

The-bang bang controller was straight forward, as we only had to specify 3 cases: if the robot is too far from the bandcenter, make it turn towards the wall; if the robot is too close to the bandcenter, make it turn away from the wall; in all other cases, make it go straight. We made the robot turn by setting different speeds for each motor, either MOTOR_HIGH or MOTOR_LOW. The challenges that we faced when we implemented this controller included turns that were too tight, resulting in the robot often hitting the wall during turns and the robot not turning quickly enough when reaching a corner. These issues were fixed by increasing the bandcenter and setting the right (side) motor to go backwards when turning at a corner. This gives the model enough space during turns and enough time during corner turns, as it stays stationary when turning to the right path. We did not need to change the bandwidth.

For the p-controller, a problem that we almost immediately encountered was the EV3 going too fast when setting the speed corrections proportional to the error. We found out that this problem occurred when the sensor read the max integer value during turns, which makes the robot go too fast. We fixed the issue during the calculation of the speed correction in a separate function called “calculateSpeed”, which takes in the distance error as input and outputs its corresponding correction. After multiplying the gain constant with the error, we checked if the speed correction was too high according to an arbitrary max speed. If it is, we set it at the max speed. We also decided to set speed corrections depending on the error, because it is more precise and intuitive. We also found the p-controller a lot less stable during its runs and compensated in the code with extra cases. When the error is within the bandwidth, the robot goes straight. When the error is above 0 (too close to the wall), 2 different cases occur: if it is too close to the wall, the robot turns at the same speed on both motors, but in different directions (forward/backward) to prevent forming an arc and bumping the wall in the process. If it is not too close to the wall, it turns away normally. If the error is negative (far from wall), it turns towards the wall normally because it does not need to compensate for corners. Finally, when the sensor detects large values (very negative error), it moves forward at a slower speed, so that the robot has time to adjust its trajectory. All speed corrections in all cases are calculated using the “calculateSpeed” function. We increased the previous bandcenter value to 35 so that there was less risk to hit the wall. We did not have to change the bandwidth as well. The p-type constant was tuned according to in-class slides: when the robot would sometimes go off-path, we increased the constant (overdamped); when the robot would stay too close to the wall, we reduced the constant (underdamped). We also had to adjust the constant for each case until it was stable.

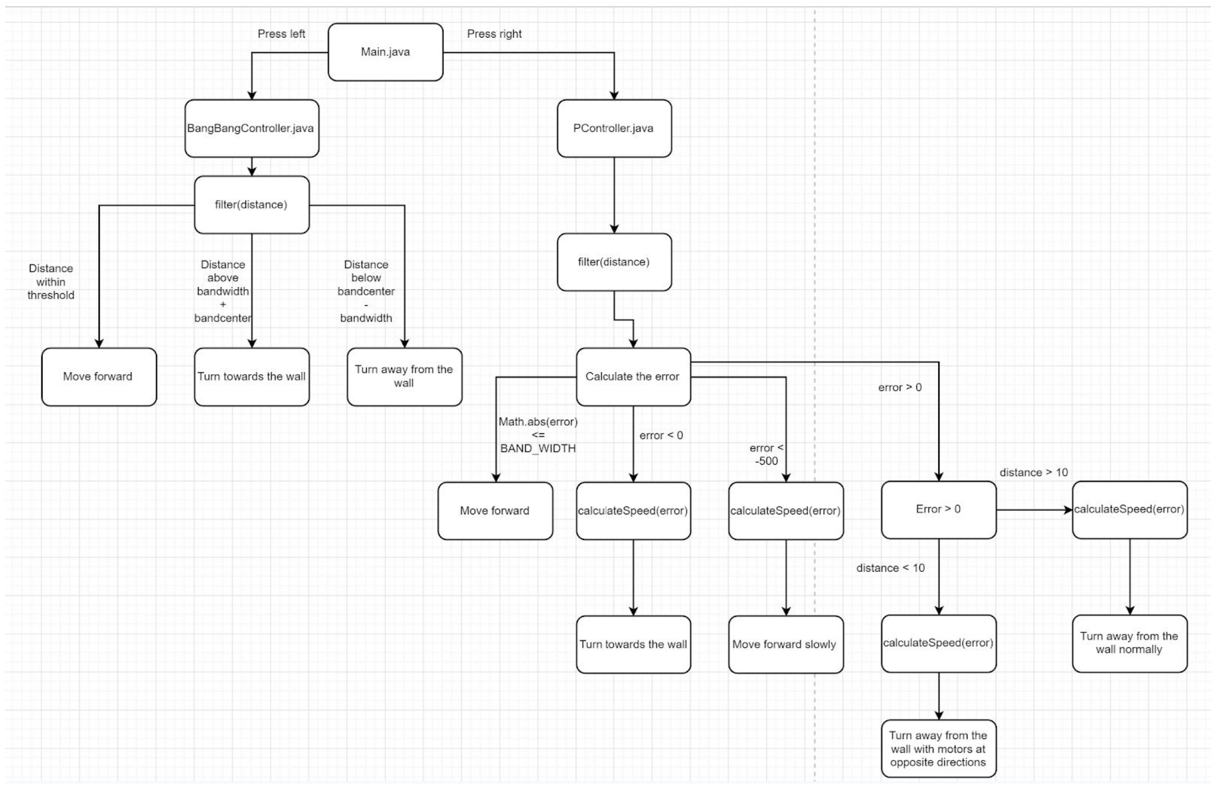


Figure 1.3. Software flowchart (PController & Bangbang)

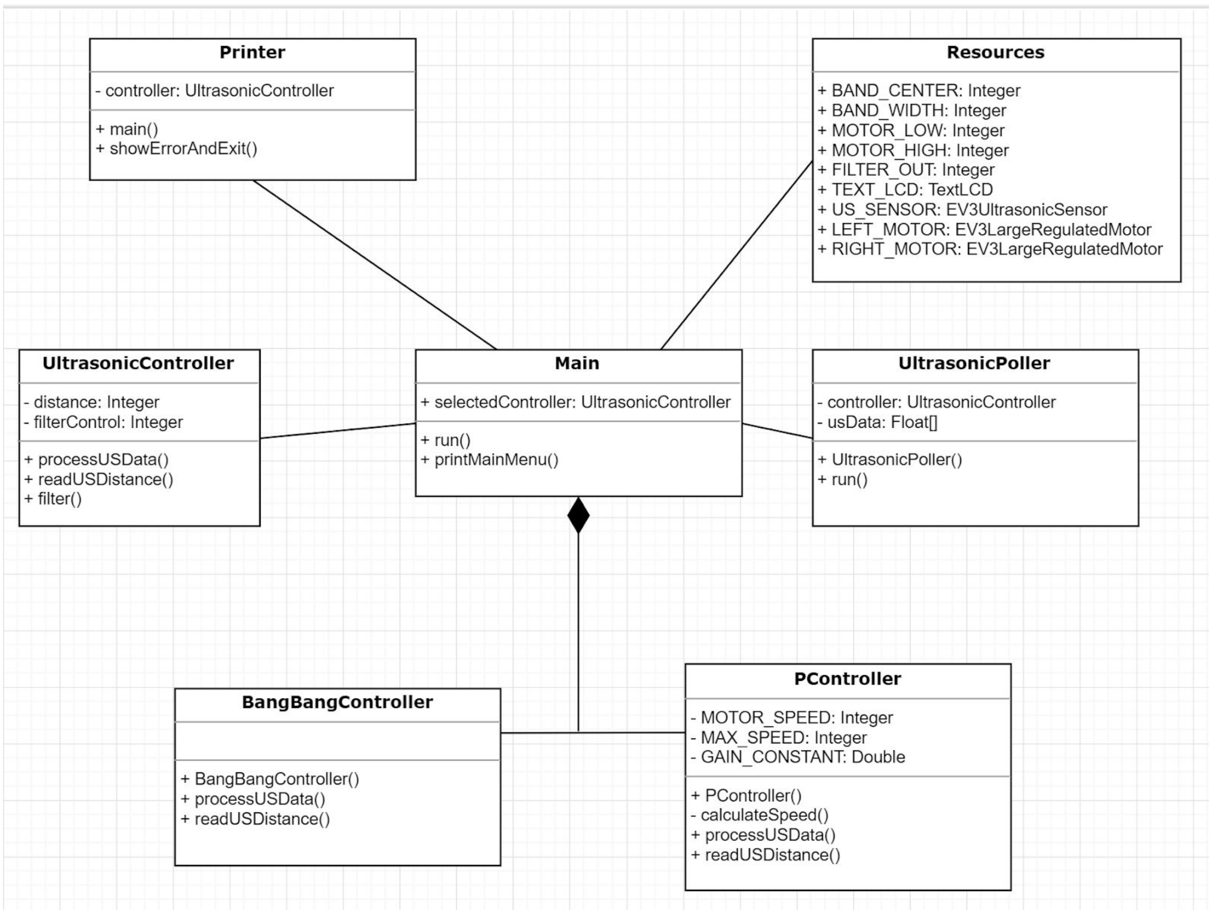


Figure 1.4. Software class diagram (PController & Bangbang)

Workflow

To complete the lab, we first built the hardware portion and adjusted the sensor placement using test runs as stated previously. The test runs followed the bang-bang controller method, as it was easier and faster to code. We then coded the p-controller method, which required a lot more testing, not only for the gain constant but also for global variables, such as the bandcenter and thread sleep time. Each time a global variable was changed, we tested 3 laps on each method (bang-bang and p-controller) because it affected both. Also, we tested our prototype each time a few lines of code were changed to make it easier to detect errors.

Test data

Fig. 2.1. Test data for values below and above the demo P-type constant

P-type constant (Demo: 8)	Band Center	Oscillations	Successful lap
Below demo value (4)	Far from band center, too close during turns	Few oscillations, increased oscillations during turns	No
Above demo value (12)	Far from band center, too close during turns	Random oscillations, increased oscillations at fast speed during turns	No

Fig. 2.2. Bang-bang controller test runs

Run #	Band Center	Oscillations	Successful lap
1	Constantly far from the wall during straight lines, very close from the wall during turns	Increased oscillations during turns, rare oscillations during straight lines	Yes
2	Constantly far from the wall, gets closer near turns, very close to the wall after the turn	Slight oscillations before a turn, frequent oscillations after a turn, none during straight lines	No
3	Constantly far from the wall, gets closer near turns, very close to the wall after the turn	Increased oscillations after a turn to correct path, increased oscillations before a turn, none during straight lines	Yes

Fig. 2.3. P-controller test runs

Run #	Band Center	Oscillations	Successful lap
1	Constant distance from the wall, gets closer before turn, really close after turn	None during straight lines and during the turn, few before a turn, many after a turn	Yes
2	Constant distance from the wall, really close after a turn	Few during straight line, many after the turn	No
3	Constant distance from the wall, gets a bit closer during turns	None during straight lines, few after a turn, none during a turn	Yes

Test analysis

What happens when your P-type constant is different from the one used in the demo?

Observed problems during the 2 laps were an increase in oscillations during and after turns. Other observed behaviors when we tested our robot on our own included the EV3 going off-path or getting too close to the wall during turns. There were also random oscillations when it followed a straight wall and the overall speed was not optimal.

How much does your robot oscillate around the band center?

Our robot oscillates rarely when it follows a straight wall. During outer turns, the EV3 follows the same arc until it is too close to the wall, where it starts oscillating a lot to try and get away from the wall. The same behavior can be observed in both controllers, except that the bang-bang controller oscillates less after a turn. However, the robot is more stable during corner turns in both controllers, where it never needs to correct its pathing by oscillating. This occurs because corner turns are safer: wheels are going opposite directions, which stops the robot going forward while it corrects its direction.

Did it ever exceed the bandwidth? If so, by how much?

For the bang-bang controller, it rarely exceeded the bandwidth when following straight walls and if it did, it was in the range of [0, 5] cm. During turns, it would more frequently exceed it by around [5, 10] cm. On the other hand, the P-controller exceeded the bandwidth by [0, 10] cm when travelling along straight walls, where the value would increase incrementally when approaching a turn. At turns, the value would increase to around [12, 21] cm very frequently (often tight turns).

Describe how this occurs qualitatively for each controller.

For the bang-bang controller, oscillations are consistently low when following a straight wall and slightly increase before a turn. The robot also needs a few corrections after a turn, but it usually fixes its pathing in max 3 oscillations. For the P-controller method, the oscillations are faster and sharper after a turn, which makes it a bit unstable. This results in more swaying than the other controller as it tries to get away from the wall, while still swaying towards it. However, its corner turns and straight-line trajectory is as stable as the bang-bang controller.

Observations and conclusions

Based on your analysis, which controller would you use and why?

According to the test data, we would use the bang-bang controller because it is a lot more stable all around. It has the same advantages as the p-controller while performing better: it keeps a stable path when following straight walls and uses motors going at opposite directions during corner turns, just like the p-controller. It also oscillates less after making an outward turn, making it less risky to hit the wall. Another major advantage is the resources needed to implement the bang-bang controller: the code was easy to implement and was not time-consuming. We did not have to run multiple tests and change values, as the distance corrections were not proportional to the error. We did not have to test every change in the gain constant, set a maximum speed or compensate for more cases.

Does the ultrasonic sensor produce false positives (detection of non-existent objects) and/or false negatives (failure to detect objects)? How frequent were they? Were they filtered?

The sensor was stable overall, but it's readings would sometimes jump around, which causes the robot to oscillate when navigating straight lines. However, it did not happen often. The issue was more apparent and frequent before and after a turn. We did not have to make changes in the bang-bang controller because the robot would usually correct it’s pathing quickly, but we had to set a maximum speed for the p-controller method, as the robot would go out of control when false values were read. More specifically, when a high value was falsely displayed, it would cause the EV3 to go extremely fast because the speed correction is proportional to the error. This causes it to go off the path or crash into a wall.

Further improvements

What software improvements could you make to address the ultrasonic sensor errors? Give 3 examples.

1. We can reduce the speed of the robot when false values are read (i.e. sudden big value when travelling in a straight line) to give it more time to correct its pathing.
2. We can vary the thread sleeping time according to the robot’s distance to the wall: if its distance is far from the wall, we do not need to track a lot of values because there is no apparent danger; if its distance is close to the wall, we need to track more values to make sure it does not collide with the wall.
3. We could use a function to predict the next value so that the speed of the robot can be reduced if an expected value is not met during its trajectory to give it more time to correct its path (i.e. when turning away from the wall, it will expect a close

distance value when it reaches the wall, but a false small distance causes it to go off-path).

What hardware improvements could you make to improve the controller performance? Give 3 examples.

1. We could improve the cable management of the robot and use pieces to hold them tight, as the sensor cable sometimes pull it in a different direction.
2. We could reduce the distance between motors to increase the robot’s agility; it could make smaller turns, which will make it more precise.
3. We could replace the ball support with another wheel to add more control, as the ball is imprecise and does not provide friction with the ground. A wheel would guide the robot in its correct path more consistently.

What other controller types could be used in place of the Bang-Bang or P-type?

We could use a PID controller, which is similar to the P-type controller. The PID controller will continuously calculate the distance error and will apply the correction using proportional, integral and derivative terms. Using more terms could create a more accurate and optimized model for the robot. We could also use the PI controller, which uses proportional and integral only. It is, therefore, less complex than the PID controller, while still providing the use of integrals, which eliminates the offset.