Engineering Notebook

TEAM 271

ENED 1120 – 021

Dr. Cedrick Kwuimy

Project 5: Autonomous Record Retriever

Project date: Feb 2, 2020 to Feb 9, 2020

**Team members:**

John Cummings | cummijh

Nathan Damas | damasnj

Bao Huynh | huynhlbg

Triet Pham | phamt7

**Goal:**

Bla bla bla.

Table of Contents

[Table of Figures ii](#_Toc34489010)

[Meeting 1: Breaking down project 5 1](#_Toc34489011)

[Meeting 2: Navigation system 7](#_Toc34489012)

[Meeting 3: Pickup system 13](#_Toc34489013)

[Works Cited 16](#_Toc34489014)

# Table of Figures

[Figure 1: Barcode types and locations 1](#_Toc34489030)

[Figure 2: Demonstration arena overview 2](#_Toc34489031)

[Figure 3: Visualization of IPS System 3](#_Toc34489032)

[Figure 4: Project 5's work breakdown structure 5](#_Toc34489033)

[Figure 5: Project 5’s Work precedence network 6](#_Toc34489034)

[Figure 6: Project 5's Gantt Chart 6](#_Toc34489035)

[Figure 7: Conceptual design for robot movement system 7](#_Toc34489036)

[Figure 8: Prototype of robot movement system 8](#_Toc34489037)

[Figure 9: Flowchart for PID going straight algorithm 9](#_Toc34489038)

[Figure 10: Flowchart for turning 180 algorithm v.1 10](#_Toc34489039)

[Figure 11: Flowchart for turning 180 algorithm v.2 11](#_Toc34489040)

[Figure 12: Test data 1 going straight 12](#_Toc34489041)

[Figure 13: Test data 2 going straight 12](#_Toc34489042)

[Figure 14: Test data 3 going straight 12](#_Toc34489043)

[Figure 15: Conveyor belt frame + pickup arm for the lift system 14](#_Toc34489044)

[Figure 16: Final design for robot's lifting mechanism 15](#_Toc34489045)

# Meeting 1: Breaking down project 5

*Feb 12, 2020  
Langsam Library, 4pm – 6pm*

***Attendance:*** *Everyone*

|  |  |
| --- | --- |
| Task | Reflection |
| 1. Read and understand project 5’s description 2. Breaking down project 5’s needed components 3. Write test plan and document into a specification review file 4. Create Gantt chart for project 5’s management | 1. We summarized project’s requirements in a more understandable way and also highlight parts that we think need more clarification ([details below](#_1._Understanding_project)) 2. We broke down project 5’s robots into 5 sub-components ([details below](#_2._Sub-components:)) 3. We wrote repetition test plans for each component and make a specification review file ([details below](#_3._Testing_plans:)) 4. We created work breakdown structure, work precedence network, and Gantt chart for project 5 ([details below](#_4._Project_management:)) |

**Details:**

## 1. Understanding project 5

We managed to give a concise description of the robot’s purpose:

**The robot has one job: Given a barcode input, the robot must scan the whole arena until it picks up the box with the right bar code**

Additional clarification:

1. **What is a barcode?**

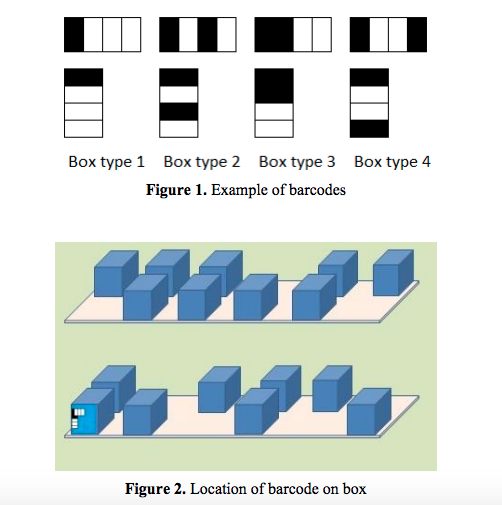
****

Figure 1: Barcode types and locations

There will be 4 types of barcodes (meaning either there are duplicate barcodes, or somebox will have no code - **must ask more on RFAI**). Then scanning techniques is used to make sure that the box the robot will choose match with the input barcode

1. **What will the arena look like?**

This is the map of the final arena:

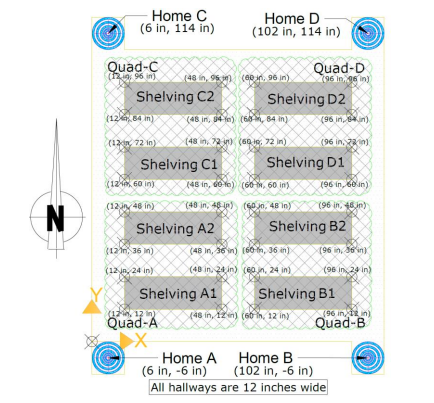


Figure 2: Demonstration arena overview

There are only shelves of boxes. No walls, no painted lines. Point A, C, D (not B) are Bluetooth transmission beacons that will help with robot’s navigation (more details below). Each shelf can contain a maximum of 12 boxes = 6 boxes \* 2 row, but reality can be less than that (meaning that some box spot will be deliberately left empty).

1. **How will the searching process be monitored?**

There are 4 zones: A, B, C, D. At the demonstration, the coordinators will choose a starting zone for our robot (we can’t choose our starting place). Also, the robot must begin its searching in this order: **Zone A → B → C → D → A**. Meaning that, if we get assigned zone C as our starting point, we must first begin by searching zone C, then Zone D, then A, then B.

(The starting position is said to be RANDOM, but we are not sure how random is that: will it always be in the middle of a random zone, near a random shelf, or is it completely random and unpredictable? - ask more in RFAI)

Some more specific restrictions include:

* If the robot starts at zone A or D, it must first turn right into the hallway, and then search the shelf on its left first
* If the robot starts at zone C or B, it must first turn left into the hallway, and then search the shelf on its right first
* The robot must not touch any other obstacles (like random human walking in hallways, or other robot). And must not touch other boxes besides the correct one.

1. **How will the robot navigate?**

Using wheels and motors.

The system to help the robot determine where it is and where it has gone is called **Indoor PS** (or IPS or Indoor Positioning System). The way it works is: The robot will receive Bluetooth signal from point A, C, D and use that to **triangulate** its position. Mathematically, the strength of signal received will be processed to give us the robot’s distance to point A, C, and D and then we can determine its location on the grid.

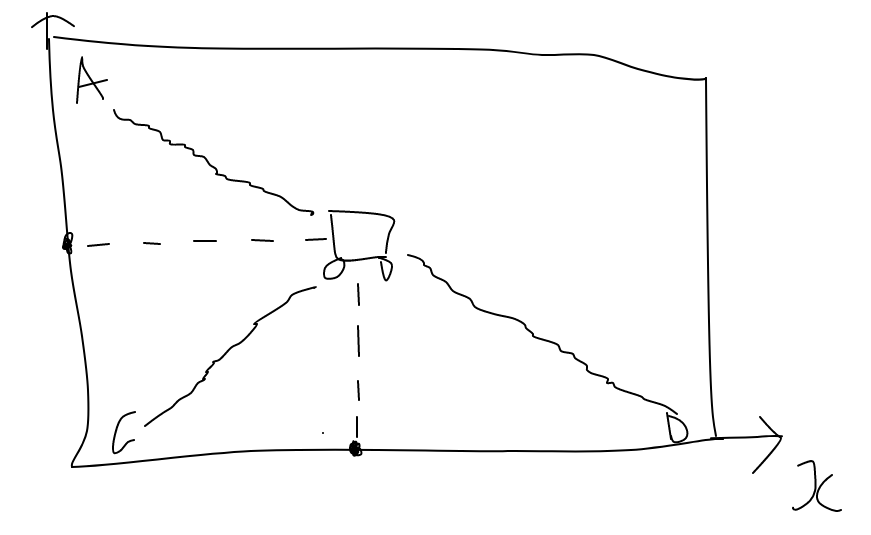


Figure 3: Visualization of IPS System

1. **How will the box look like?**

Dimension: Height: 6 in. ± 1.0 in.

Width: 3.785 in. ± 0.5 in.

Lifting Handle: A rigid handle approximately ½ inch wide is located approximately ½ inch above the box, centered and spanning the box width.

Weight: < 200 grams

Other Features: A small magnet glued on the inside center face of the box

1. **What to do after the robot has chosen the right box?**

Robot must then return to starting position to drop off and re-position to RECEIVE ANOTHER BARCODE

1. **What to do after the robot has chosen the wrong box?**

We don’t know

## 2. Sub-components:

After understanding about the robot’s purpose, we decided the robot should have the following four components:

1. Navigation system: The robot must first the able to move smoothly in the arena, so a navigation with wheels or treads is necessary
2. Barcode scanning system: The robot must be able to read the barcode correctly to get the right box
3. Pick up system: A good pick up system helps the robot retrieve a box surely (not dropping the box half-way) and in a timely manner. The system also manages dropping off the box when going back to the HOME location
4. Storage system: A storage system is necessary to ensure that the box is safe when transporting between location no matter how far. The system also ensures if anything happens, the box’s content shall remain intact
5. Localization system: A localization system with Bluetooth receiver help the robot triangulate its position to navigate the arena.

## 3. Testing plans:

Next, we developed testing plans for each function with specific numbers and parameters to make sure the above components work properly

1. Navigation system

* Minimum speed: 1 ft/s
* Be able to go forward perfectly straight – for 5 times in a row; will test with different distances from 1 to 10ft
* Be able to go backward perfectly straight – for 5 times in a row; will test with different distances from 1 to 10ft
* Be able to turn precisely with a displacement less than 3 inches – for 5 times in a row; will test with different angle from 45 to 270 degrees
* Each test above will be replicated on difference surface: carpet, paper, and tile

1. Barcode scanning system

* Barcode scanning maximum speed: less than 10s
* Be able to determine the right barcode type out of the 4 types – for 15 times in a row
* Be able to determine additional made-up barcodes as INVALID – for 15 times in a row

1. Pick up system:

* Be able to pick up maximum of 250g
* Be able to pick up boxes without dropping half-way successfully – for 10 times in a row, with boxes of different weights
* Be able to pick up boxes and put correctly in storage system and still able to move a small distance after that successfully – for 10 times in row; maximum time for this test must be less 10s for each repetition

1. Storage system

* Be able to carry object of maximum 250g without making the robot malfunction
* Be able to carry the object safely through a minimum distance 5 ft successfully – for 10 times in a row; will test for different speed: 1ft/s to 3 ft/s
* Be able to keep the object safe while robot spins at high speed for 10s

1. Localization

* Be able to display the correct (x, y) position of the robot – for 20 times in a row
* Be able to navigate between two coordinates precisely (with error < 5%) – for 15 times in a row; will test for different distances, including edge cases

## 4. Project management:

Based on the functional break down of the robot, we began project 5’s management with a work breakdown structure. We identified that the project will have three important phases:

I. Research phase: There are many important questions to explore to choose the optimal design for our robot: What factor contributes to the robot’s navigation problems (if any)? How does real-life barcode reader work and can we re-implement them here? What kind of picking machinery/designs is best for picking up boxes and how to not accidentally pick up the wrong box?, etc.

II. Building and testing phases: This is when all the research has been done, decision matrixes has been completed and we start to build the actual robot. We will build and test each component carefully before integrating the next components.

III. Post-demonstration phase: This is when the robot has finished and demonstration day has passed. The work left for us will include: Prepare for presentation, re-format engineering notebook, and write reports.

With these ideas in mind, below are the images of our Work breakdown structure, Work precedence network, and Gantt Chart. We estimated the project will be completed in 9 weeks:

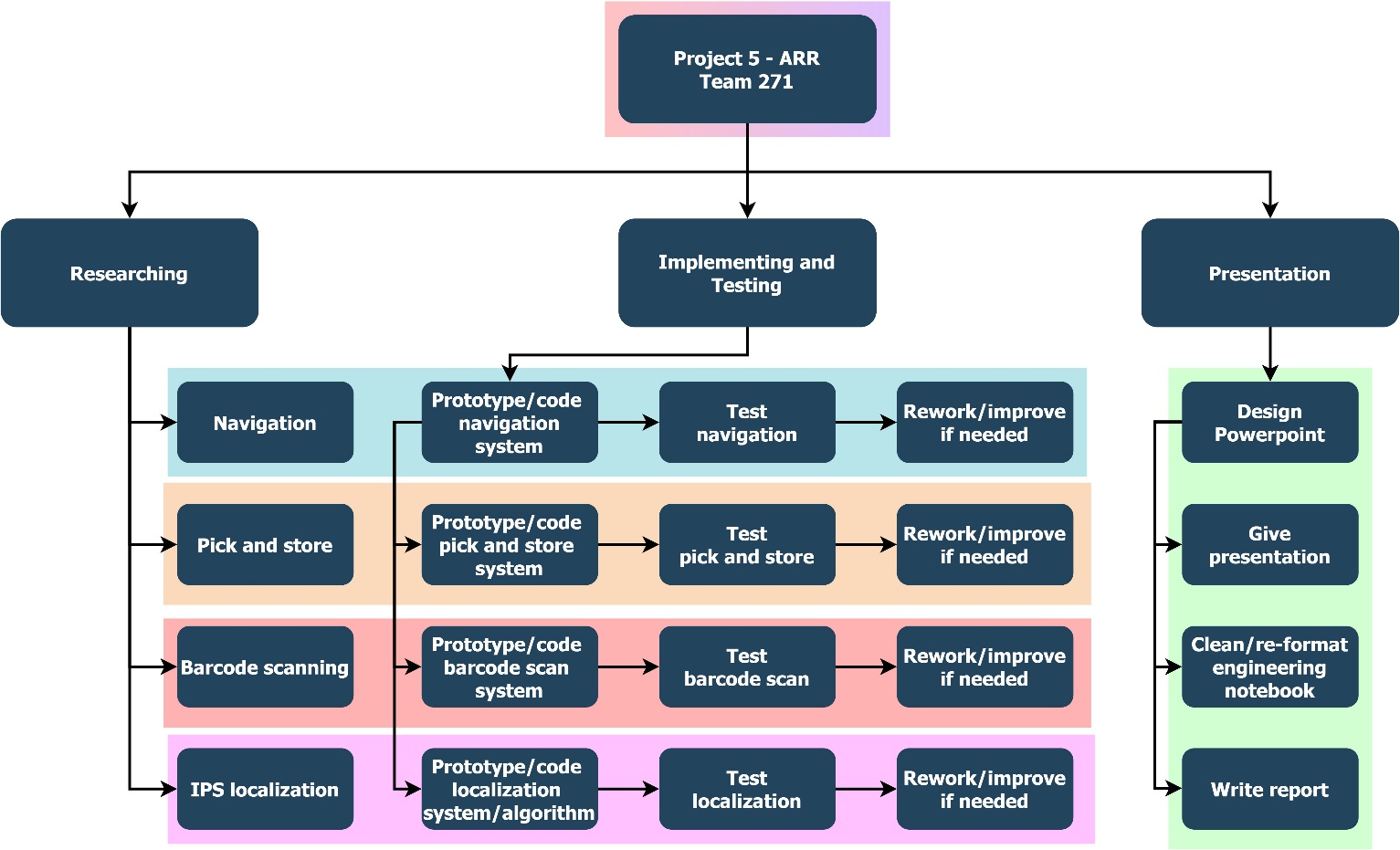


Figure 4: Project 5's work breakdown structure

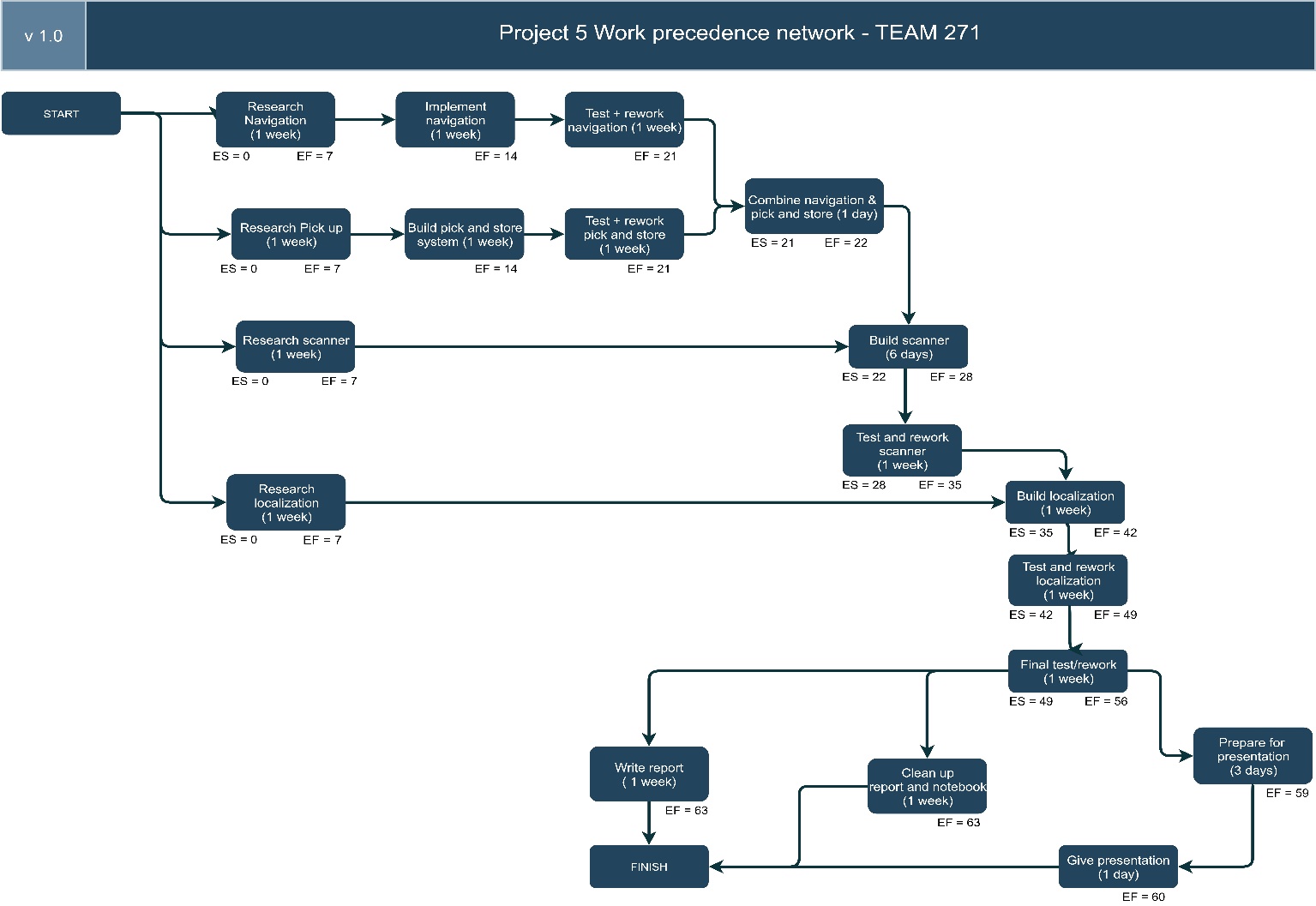


Figure 5: Project 5’s Work precedence network

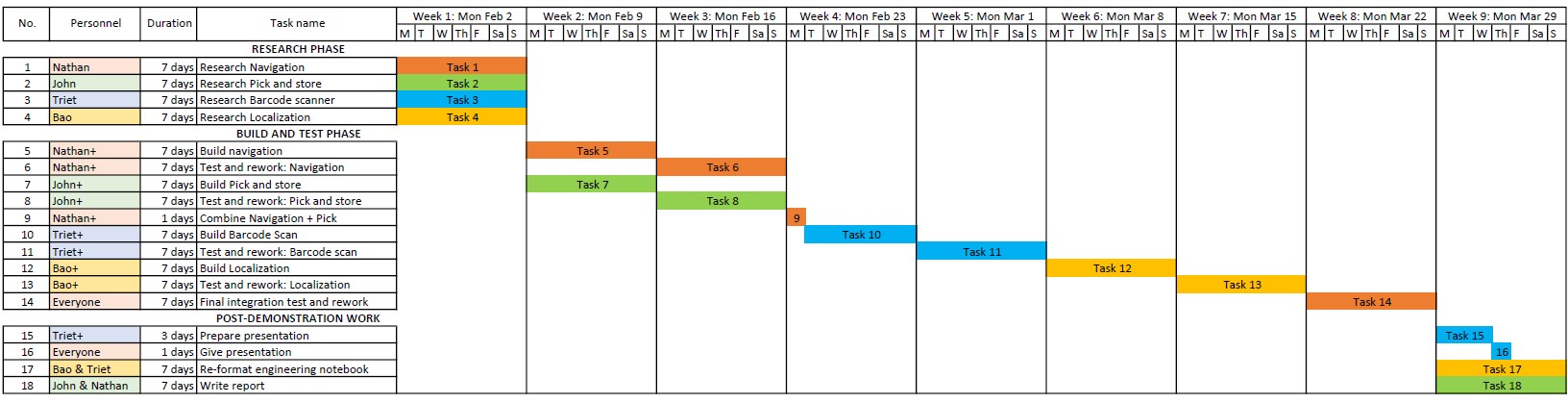


Figure 6: Project 5's Gantt Chart

# Meeting 2: Navigation system

*Feb 14, 2020  
Langsam Library, 3:30pm – 6:30pm****Attendance:*** *Everyone*

|  |  |
| --- | --- |
| Task | Reflection |
| 1. Brainstorm, choose idea, and prototype the robot’s navigation/movement system 2. Implement a program to make robot go forward and backward 3. Implement a program to make robot turn 180 degree 4. Test for robot speed | 1. We built a robot with 2 main wheels in front and a small metal marble to support the back ([details below](#_1._Robot_prototyping)) 2. Our Python code implemented a PID algorithm with gyro sensor to help redirect the robot if it veers ([details below](#_2._Going_straight:)) 3. Using gyro sensor readings, we programmed the robot to turn until a desired angle is reached ([details below](#_3._Turning_180)) 4. We tested with 20 trials, on tile and paper surface; the robot speed is determined to be 10cm/s (going forward) and 10.7cm/s (going backward) ([details below](#_4._Determine_the)) |

**Details:**

## 1. Robot prototyping:

After reading the descriptions of project 5, we decided to use **wheels** as the mean of transportation, which is both easy to implement and control, while also agile enough to allow robot to dodge obstacles in the arena.

We also discussed and decided to go with 2 wheels because it has less variability. Having 4 wheels would mean gears are needed to connect the front and back part, but gears have big variability due to popping in and out.

However, upon building the two big wheels into the EV3 bricks and the two motors, we saw that our robot is falling downward at the back part because the two wheels assembled were at the front. Therefore, added a metal ball at the back to balance out the two wheels at the front, creating a strong triangular structure.

The final design of our robot navigation system is described in the images below:

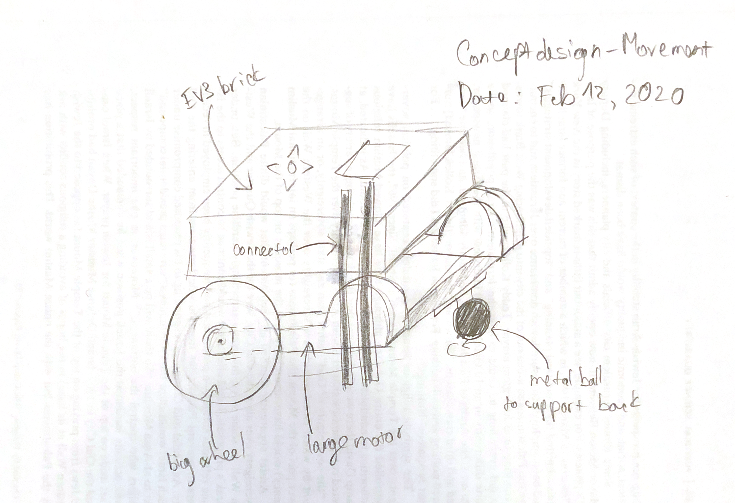


Figure 7: Conceptual design for robot movement system

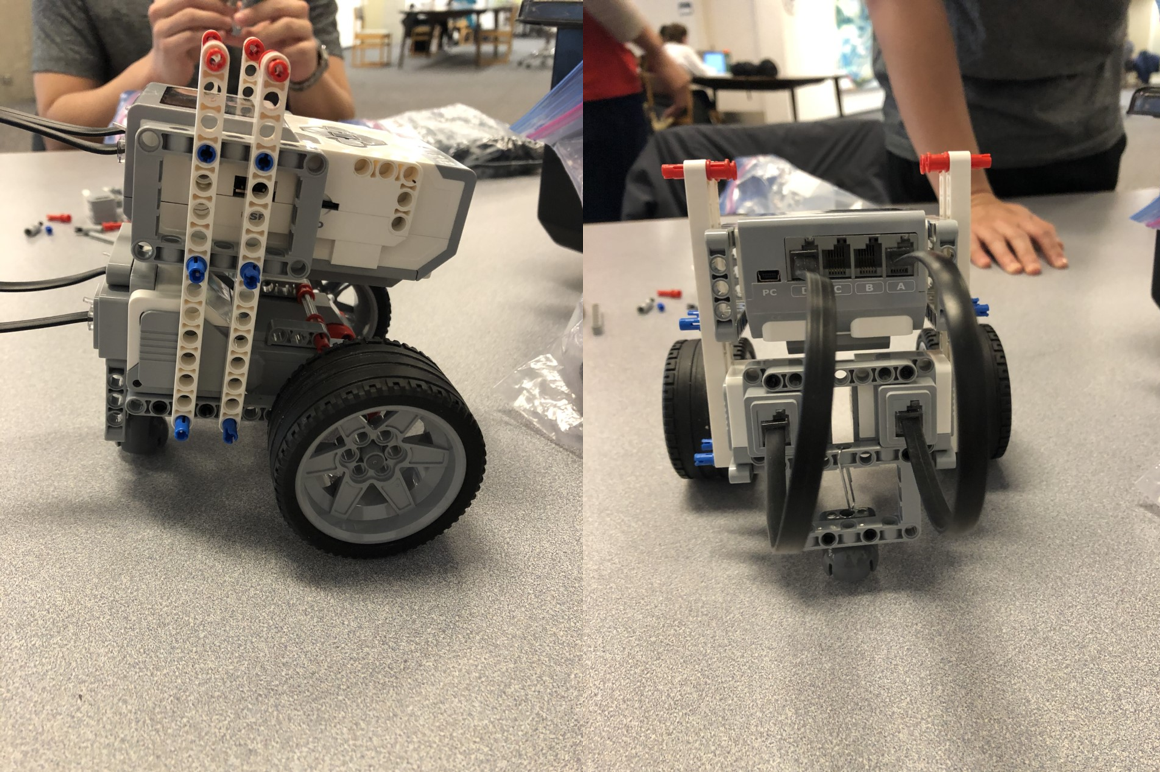


Figure 8: Prototype of robot movement system

## 2. Going straight:

We first let the robot move forward and backward without any sensor assistance. However, the robot did not go very straight, and started to veered off-track after about 1m. We figured it could have been the inherent faults in the EV3 motors and EV3 wheels, so there was not much we can do about that.

Therefore, we decided to implement an additional gyro sensor to read the angle as the robot is moving, then implement an algorithm to tune the motors’ speed accordingly, helping redirect the robot when the angles show that the robot is going off-track. Also, the gyro sensor can help tremendously with getting the robot to turn a desired angle (180 degrees in our case).

We did online research and found that the PID algorithm (proportional–integral–derivative controller) is “a control loop mechanism employing feedback that is widely used in [..] applications requiring continuously modulated control” (Source: [PID Controller, 2020](#_Works_Cited)), which suits out need for continuous regulation of the motors’ speeds based on the gyro sensor’s readings.

After studying from online resource (Source: [EV3 Gyro Sensor + PID Algorithm, 2018](#_Works_Cited)), our program is described in the following flowchart:

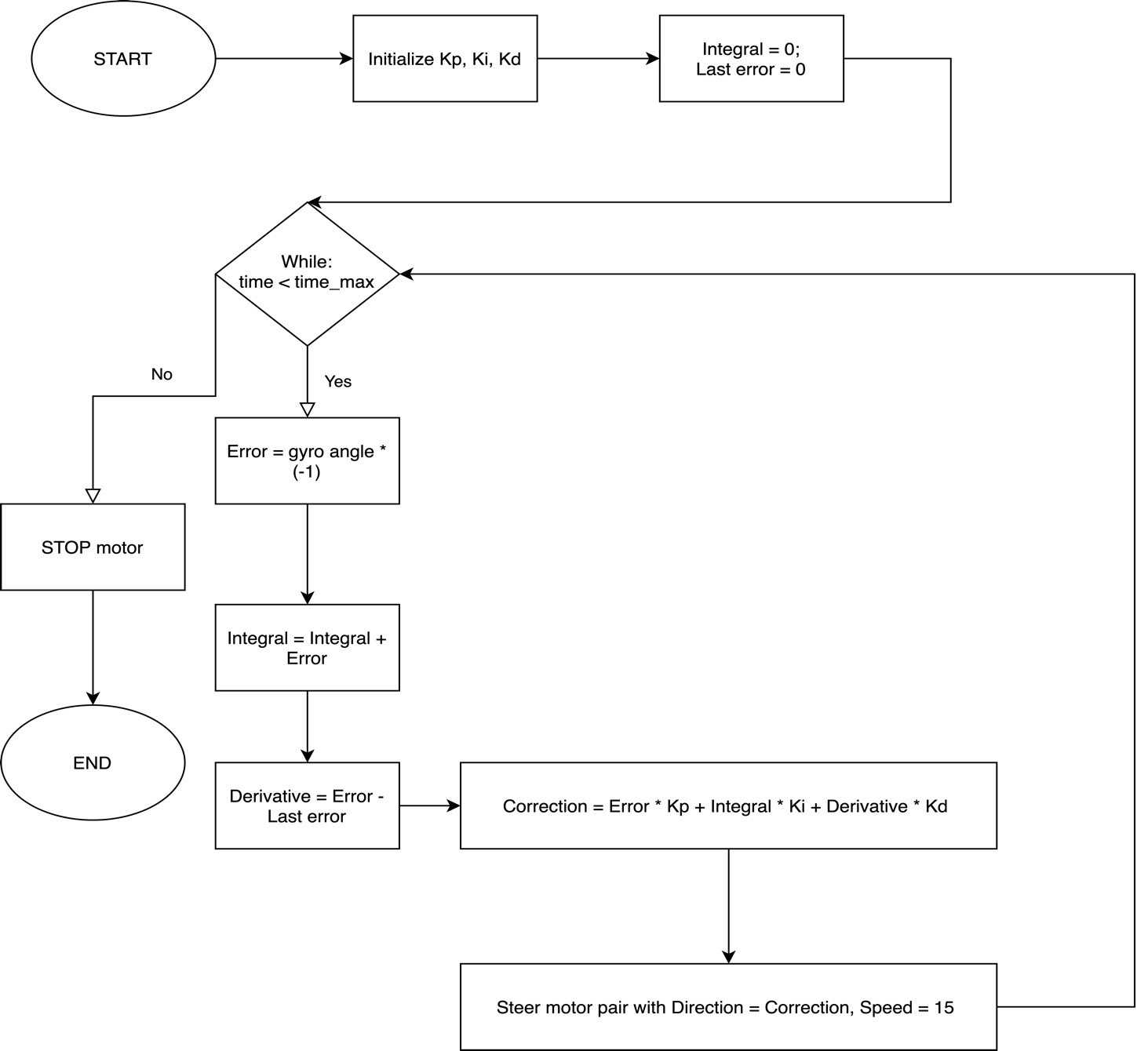


Figure 9: Flowchart for PID going straight algorithm

The movement/direction of the robot and the speed of the motor pair is controlled through a “Move Steering” function available in Python for EV3, which takes two input parameters: Direction (negative direction means moving to the left, positive direction means moving to the right, and 0 means going straight) and Speed.

The challenging part of the PID algorithm is find the suitable Kp, Ki, and Kd values that suits our robot’s weight distribution and thus making the robot go straight. After 30 minutes of trying out different set of values, we found the set of values that make the robot goes straight the most is (Kp = -0.149495; Ki = -0.0001; Kd = 0)

Our Python code is thus written as followed:

def forward\_with\_time(steer\_pair\_name, gyro\_sensor\_name, time\_sec):

   startTime = time()

   Kp = -0.0149495

   Ki = 0.0001

   Kd = 0

   integral = 0

   last\_error = 0

   while (time() - startTime <= time\_sec):

       error = gyro\_sensor\_name.angle \* (-1)

       integral += error

       derivative = error - last\_error

       correction = error \* Kp + integral \* Ki + derivative \* Kd

       last\_error = error

       steer\_pair\_name.on(correction, 15)

def main():

   steer\_pair = MoveSteering(OUTPUT\_D, OUTPUT\_A)

   gyro\_sensor = GyroSensor(INPUT\_2)

   reset\_gyro(gyro\_sensor)

   forward\_with\_time(steer\_pair, gyro\_sensor, time\_sec=9) # Forward 9 sec

## 3. Turning 180 degrees:

After enabling the robot to go straight, we continued the meeting with writing an algorithm for making the robot turn a desired angle.

Our initial algorithm is described in the following flow chart:

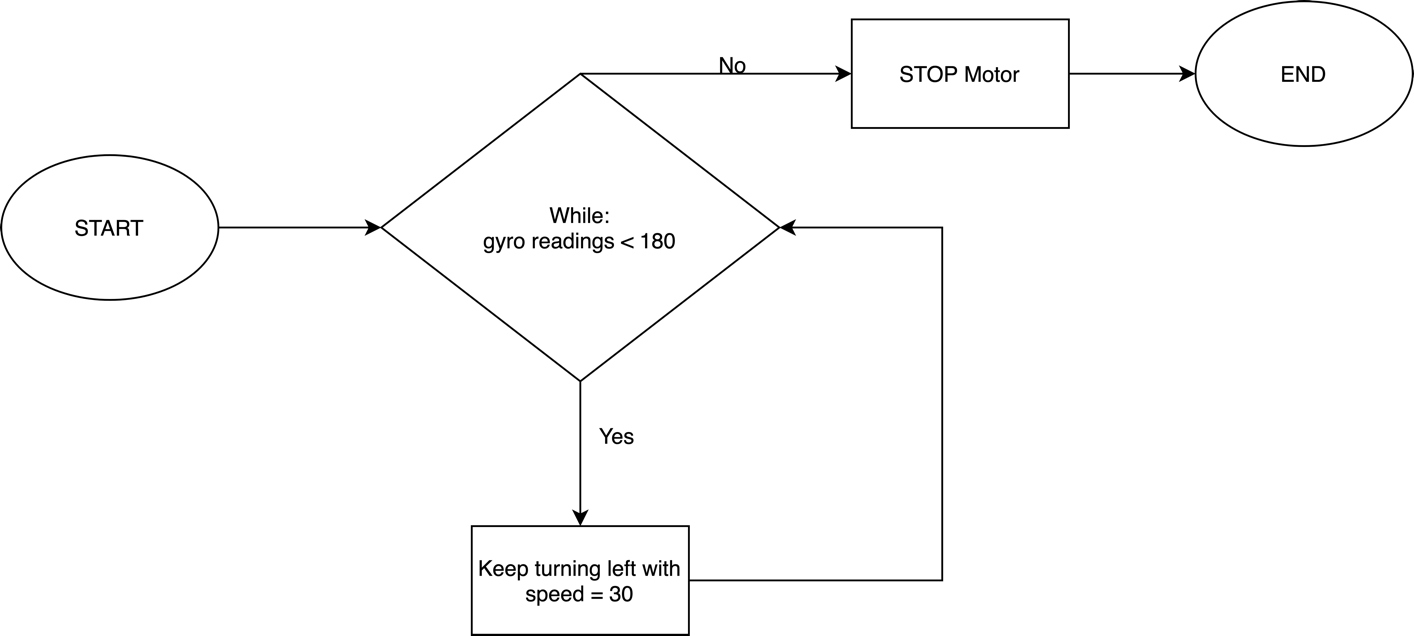


Figure 10: Flowchart for turning 180 algorithm v.1

However, we realized that the robot always had remaining acceleration from continuously turning left, making it turn farther than expected. Therefore, we think to improve our algorithm by both slowing down the robot the more it gets closer to the desired angle, and reduce the desired angle to account for residual acceleration.

Our version 2 algorithm for turning 180 degrees is as in the flowchart followed:

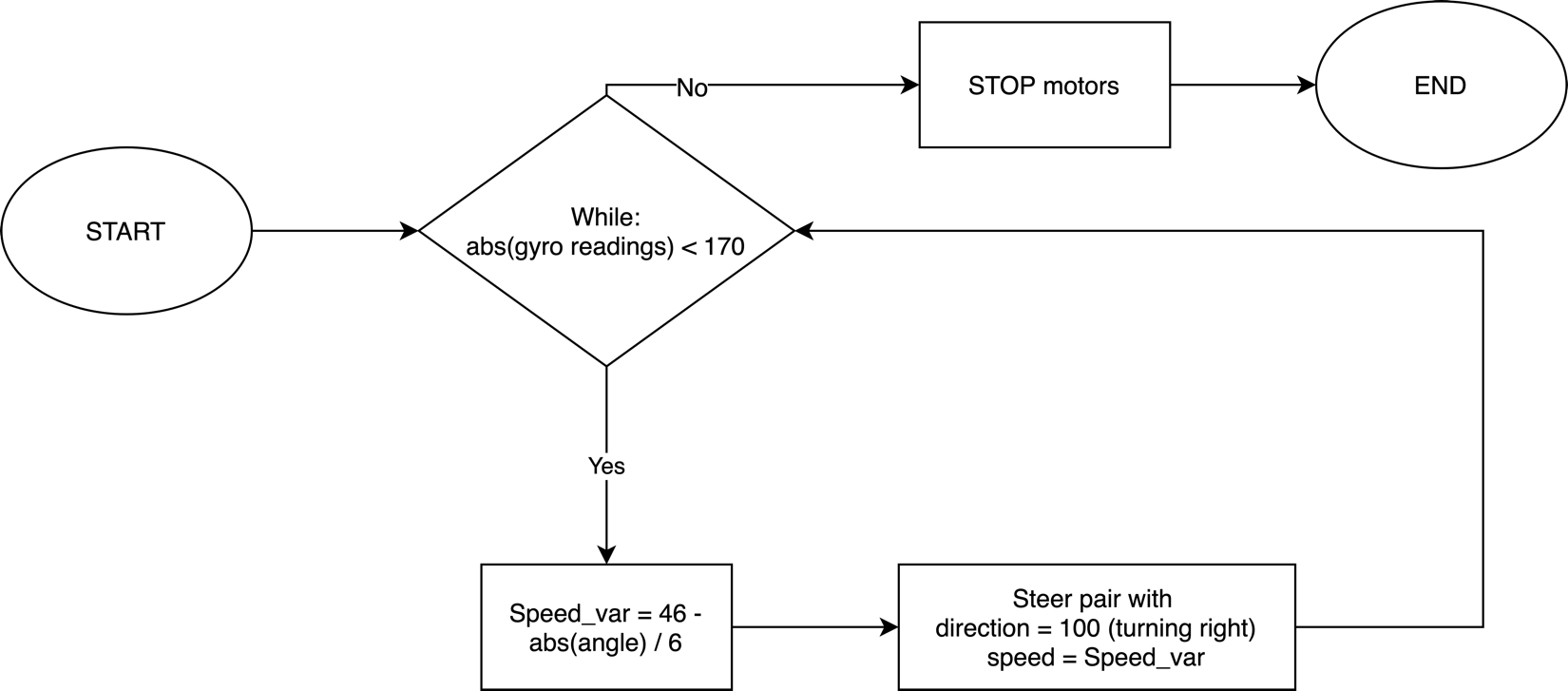


Figure 11: Flowchart for turning 180 algorithm v.2

Our Python code is written as followed:

def turn180(steer\_pair\_name, gyro\_sensor\_name):

   while True:

       angle\_positive = abs(gyro\_sensor\_name.angle)

       steer\_pair\_name.on(steering=100,

                          speed = 46 - round(angle\_positive / 6))

       if (angle\_positive >= 170):

           steer\_pair\_name.off()

           break

   return

def main():

   steer\_pair = MoveSteering(OUTPUT\_D, OUTPUT\_A)

   gyro\_sensor = GyroSensor(INPUT\_2)

   reset\_gyro(gyro\_sensor)

   turn180(steer\_pair, gyro\_sensor)

   reset\_gyro(gyro\_sensor)

## 4. Determine the robot’s speed:

After having successfully implemented the two navigation algorithms, we tested the robot on multiple surfaces to determine its speed. Our test plan included:

* Testing going forward and going backward individually: Running 5 trials forward, 5 trials backward, on both tiled floor paper surface, for different amount of times, and then measure the distances to calculate the average speed.
* Testing forward and backward together: Move forward 10s, then backward 5s, then forward 2s, then backward 7s; the final position of the robot should be the same as the start position1

Our test data is as followed:

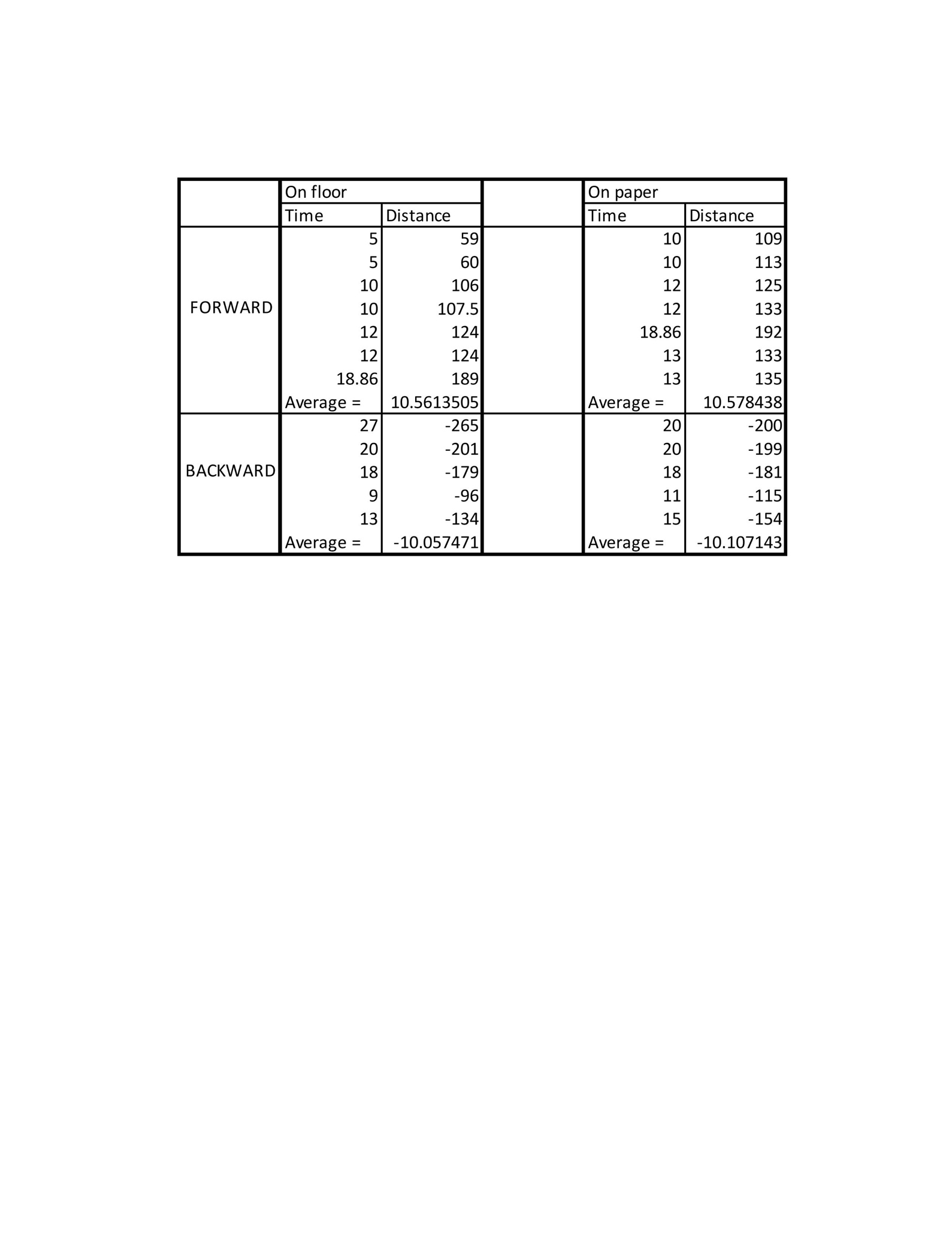
Testing going forward and going backward individually:

Figure 12: Test data 1 going straight

This test showed us that there are not much different between floor and paper surface, so from now we would only test on floor surface.

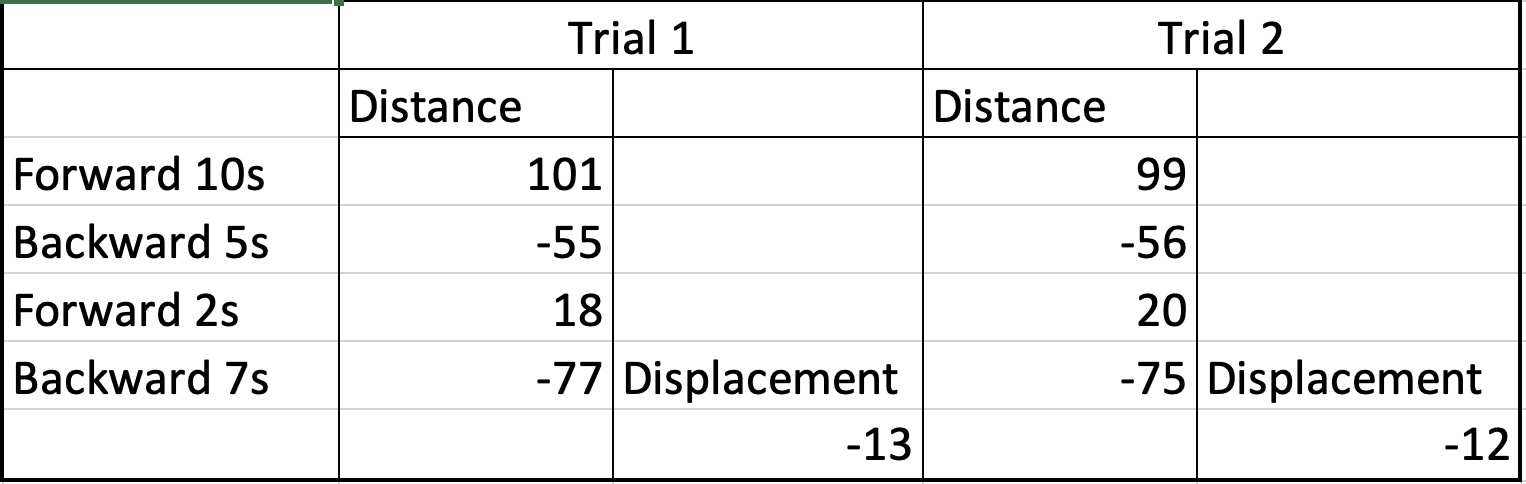
Testing going forward and backward together:

Figure 13: Test data 2 going straight

After running two trials of integration test, we realized that the backward speed was somehow faster than the forward speed, so we changed our test plan to determine how faster the backward speed was by assuming different number and run test to see if the assumption was correct. The test data is as followed:



Figure 14: Test data 3 going straight

From the three tests, we concluded that the forward speed is 10cm/s, and the backward speed is 10.7cm/s

# Meeting 3: Pickup system

*Feb 19, 2020  
Langsam Library, 4pm - 5:45pm*

***Attendance:*** *Everyone*

|  |  |
| --- | --- |
| Task | Reflection |
| 1. Critique and improve robot design from the performance of the demo  2. Deciding on a lifting mechanism for the robot  3. Building the prototype lifting system  4. Test the functionality | 1. We added support for the back, reinforced the robot’s frame on the bottom layer, and refined the turning algorithm ([details below](#_1._Reinforcing_robot’s))  2. With a decision matrix’ help, we agreed on a hook and lift system to pick the box ([details below](#_2._Building_the))  3. We built the hook system with conveyor belt and a protruding lifting arm (details below)  4. We tested the pickup system using different weights we had (details below) |

**Details:**

## 1. Reinforcing robot’s design:

Upon observing the robot’s performance at the subtask 1 demonstration day, the biggest problem we saw was the incorrect 180-turning of the robot. We observed that the robot turned more than it needed, probably because of low friction of the testing paper, so we changed our turning algorithm to have a small “desired angle”, specifically 160 degree, so that when the robot receives the STOP command when the angle has reached 160 and starts braking the motor, the remaining acceleration in the motors pushed the robot to the correct position of 180 degree.

Another detail we observed was the motor frame on the bottom layer was not rigid enough and could move sometimes. Therefore, we added support bars to reinforce the motors.

## 2. Deciding on the lift/pickup system:

The next system we aimed to build, according to our Gantt Chart, was the lift system. We brainstormed some possible ideas for this lift system and came up with decision matrix below:

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria  (higher point is more important) | Hydraulic forklift  (higher point is better) | Hook and lift (higher point is better) | Claw mechanism (higher point is better) |
| Weight power (5 points) | 5 | 2 | 4 |
| Ease of building (3 points) | 1 | 5 | 3 |
| Ease of coding (1 points) | 1 | 5 | 3 |
| Use less resources (2 points) | 1 | 5 | 4 |
| Ease of transportation after picking up box (4 points) | 4 | 3 | 1 |
| TOTAL = | 47 | 52 | 44 |

According to our decision matrix, we built the hook and lift system first, with a protruding arm to “piece” into the box’s handle and lift it up using the medium motor.

In our justification, the claw mechanism has more lifting power, but since we have only 1 medium motor, after we use the medium motor for the closing grip claw, we have no way to move the just-picked-up box around, thus the box will obstruct the robot’s path and vision.

The hydraulic forklift also has strong lifting power but is also the most difficult to implement. We discussed that it would be very hard to “sneak” the forklift under the box to lift it up, and the positioning would have to be very precise or the box would fall instead of being picked up. We concluded that forklifts in real world operate under human control, thus it not suited for an autonomous robot.

Therefore, we went with the hook and lift system, which, albeit does not have the most lifting power, has its advantages in ease of implementation and ease of transportation after picking up the box.

## 3. Building the pickup system:

Nathan started with a frame for the conveyor belt that will transport the box up, while Triet began building a strong arm to pick up the robot. A prototype for our lift system is in the image below:

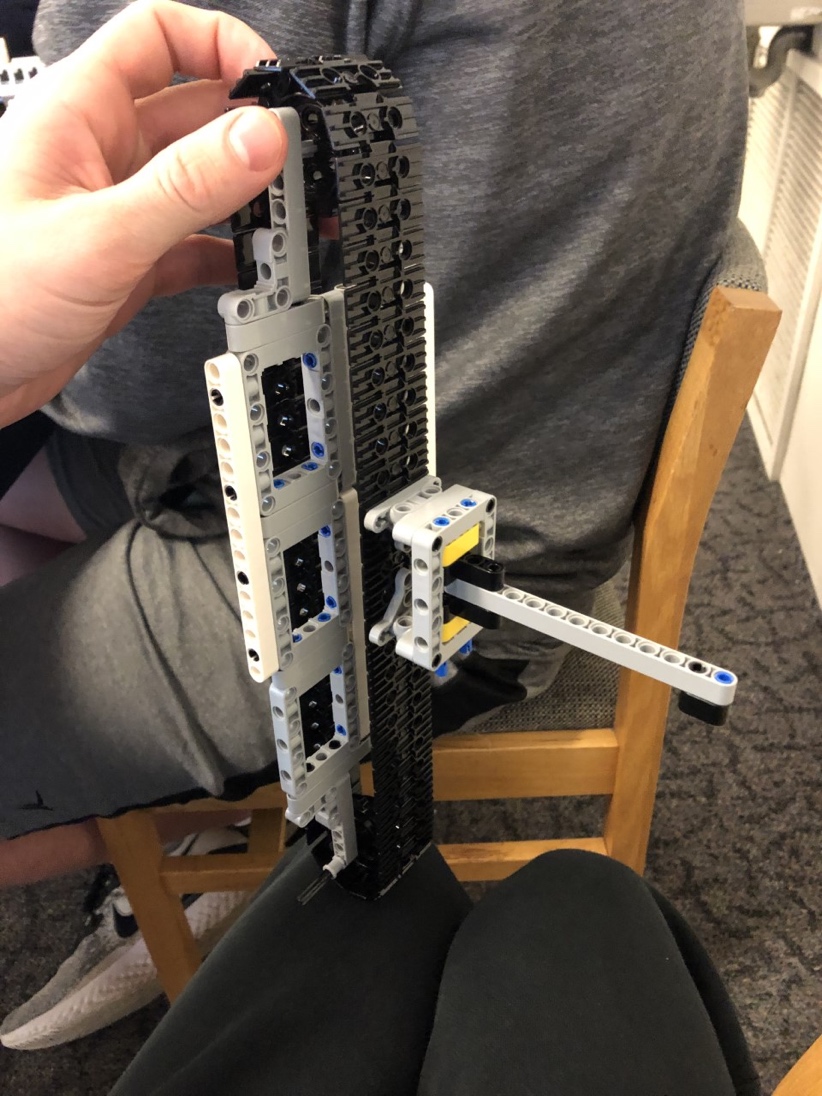


Figure 15: Conveyor belt frame + pickup arm for the lift system

Next, we incorporated this conveyor belt into the front of our robot. Upon integrating with EV3 brick, we found that the metal ball at the back was too high and pushed the robot forward, which interfered with the robot’s pickup system by making the arm leaning downward. So we redesigned the connector for the metal ball 1 “LEGO Level” lower so it fits perfectly with the robot’s weight.

Then, to make the conveyor able to spin and move the box upward after picking the box, we used a set of gears chained together with the medium motor.

The final design of our incorporated lifting system is in the following image:

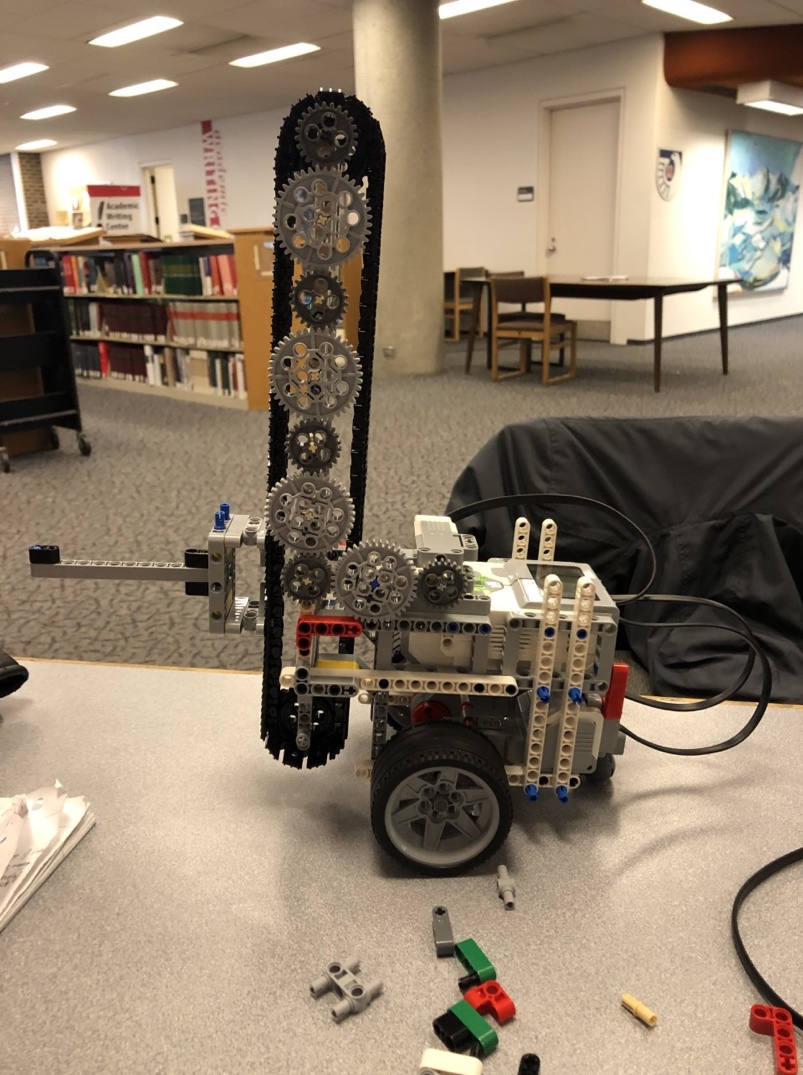


Figure 16: Final design for robot's lifting mechanism

## 4. Functionality testing:

Next, we made a simple script to test the robot’s lifting power. We used various objects to determine the maximum capability of the robot’s arm, and found that the robot was able to handle our 250g tape measure, which exceeded the weight requirement in Project 5’s Request for Proposal. Therefore, we concluded our design as successful

# Works Cited

*EV3 Gyro Sensor + PID Algorithm*. (2018, October 4). Retrieved from https://www.youtube.com/watch?v=U-LdBQ-vBkg

*PID controller*. (2020, February 26). Retrieved from https://en.wikipedia.org/wiki/PID\_controller