Engineering Notebook

TEAM 271

ENED 1120 – 021

Dr. Cedrick Kwuimy

Project 5: Autonomous Record Retriever

Project date: Feb 2, 2020 to April 26, 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](#_Toc38646487)

[Meeting 1: Breaking down project 5 – 02/12/2020 1](#_Toc38646488)

[1. Understanding project 5 1](#_Toc38646489)

[2. Sub-components: 3](#_Toc38646490)

[3. Testing plans: 4](#_Toc38646491)

[4. Project management: 5](#_Toc38646492)

[Meeting 2: Navigation system – 02/14/2020 7](#_Toc38646493)

[1. Robot prototyping: 7](#_Toc38646494)

[2. Going straight: 8](#_Toc38646495)

[3. Turning 180 degrees: 10](#_Toc38646496)

[4. Determine the robot’s speed: 11](#_Toc38646497)

[Meeting 3: Pickup system – 02/19/2020 13](#_Toc38646498)

[1. Reinforcing robot’s design: 13](#_Toc38646499)

[2. Deciding on the lift/pickup system: 13](#_Toc38646500)

[3. Building the pickup system: 14](#_Toc38646501)

[4. Functionality testing: 15](#_Toc38646502)

[Meeting 4: Ultrasonic sensor and color sensor – 02/26/2020 16](#_Toc38646503)

[1. Ultrasonic sensor: 16](#_Toc38646504)

[2. Color sensor and barcode scanner system: 16](#_Toc38646505)

[3. Extra reinforcement for added weight: 17](#_Toc38646506)

[Meeting 5: Going online and completing subtask 2 – 04/17/2020 18](#_Toc38646507)

[1. Project 5 going online and subtask 2: 18](#_Toc38646508)

[2. Navigation algorithm: 19](#_Toc38646509)

[3. Barcode scanning algorithm: 21](#_Toc38646510)

[Meeting 6: Coding the simulation and finalize subtask 2 – 04/19/2020 24](#_Toc38646511)

[1. Simulation code: 24](#_Toc38646512)

[2. Submitting videos for subtask 2: 27](#_Toc38646513)

[Meeting 7: Collision Avoidance system algorithm – 04/23/2020 28](#_Toc38646514)

[1. The “adjacent boxes” problem: 28](#_Toc38646515)

[2. Collision Avoidance system: 30](#_Toc38646516)

[Works Cited 30](#_Toc38646517)

# Table of Figures

[Figure 1: Barcode types and locations 1](#_Toc38539627)

[Figure 2: Demonstration arena overview 2](#_Toc38539628)

[Figure 3: Visualization of IPS System 3](#_Toc38539629)

[Figure 4: Project 5's work breakdown structure 5](#_Toc38539630)

[Figure 5: Project 5’s Work precedence network 6](#_Toc38539631)

[Figure 6: Project 5's Gantt Chart 6](#_Toc38539632)

[Figure 7: Conceptual design for robot movement system 7](#_Toc38539633)

[Figure 8: Prototype of robot movement system 8](#_Toc38539634)

[Figure 9: Flowchart for PID going straight algorithm 9](#_Toc38539635)

[Figure 10: Flowchart for turning 180 algorithm v.1 10](#_Toc38539636)

[Figure 11: Flowchart for turning 180 algorithm v.2 10](#_Toc38539637)

[Figure 12: Test data 1 going straight 11](#_Toc38539638)

[Figure 13: Test data 2 going straight 12](#_Toc38539639)

[Figure 14: Test data 3 going straight 12](#_Toc38539640)

[Figure 15: Conveyor belt frame + pickup arm for the lift system 14](#_Toc38539641)

[Figure 16: Final design for robot’s lifting mechanism 15](#_Toc38539642)

[Figure 17: Concept design for sensors layout 17](#_Toc38539643)

[Figure 18: Image of team 271 videoconferencing 18](#_Toc38539644)

[Figure 19: Flowchart for algorithm to navigate between 2 points 19](#_Toc38539645)

[Figure 20: Flowchart for algorithm to navigate entire arena 20](#_Toc38539646)

[Figure 21: Flowchart for algorithm to determine where to go next 21](#_Toc38539647)

[Figure 22: Illustration for backtrack-scanning algorithm 22](#_Toc38539648)

[Figure 23: Flowchart for backtrack-scanning algorithm 23](#_Toc38539649)

[Figure 24: Visual representation of team 271's simulation 24](#_Toc38539650)

[Figure 25: Illustration of the "adjacent boxes" problem 28](#_Toc38539651)

[Figure 26: Flowchart for backtrack scanning algorithm v.2 29](#_Toc38539652)

# Meeting 1: Breaking down project 5 – 02/12/2020

*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?**

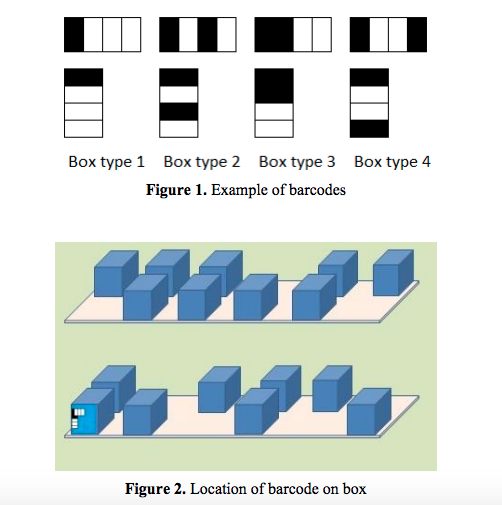
****

Figure : 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:

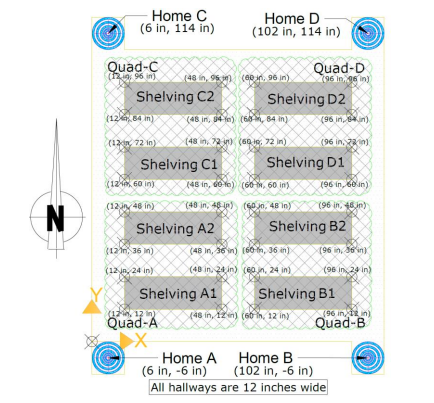


Figure : Demonstration arena overview

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.

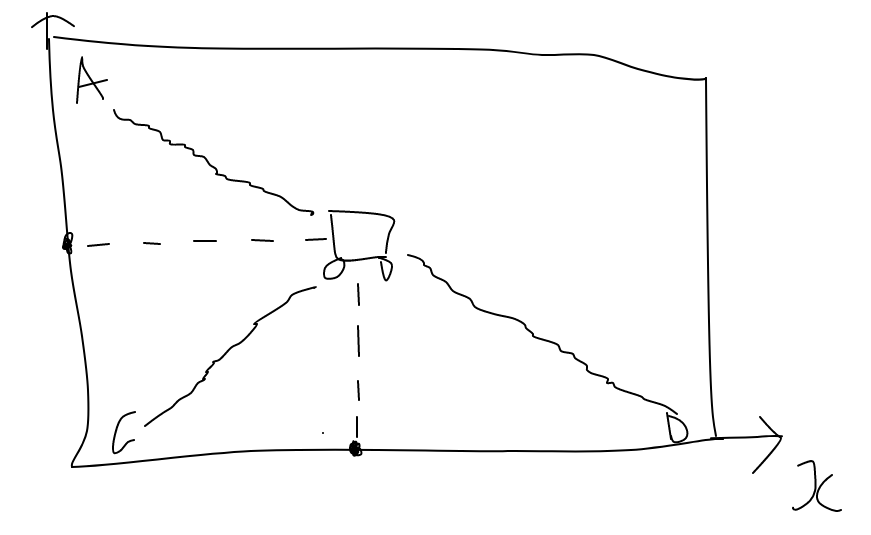


Figure : Visualization of IPS System

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:

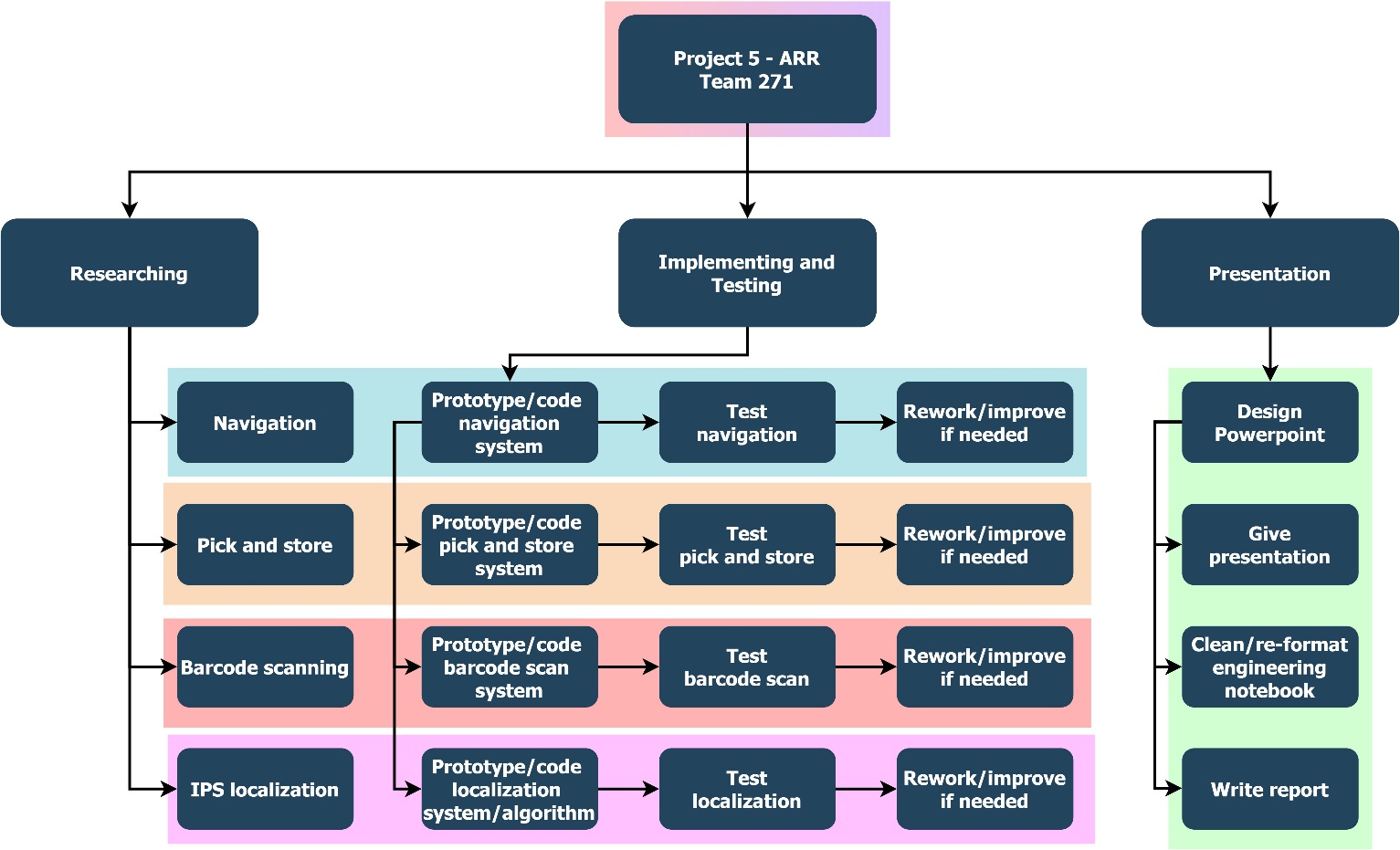


Figure : Project 5's work breakdown structure

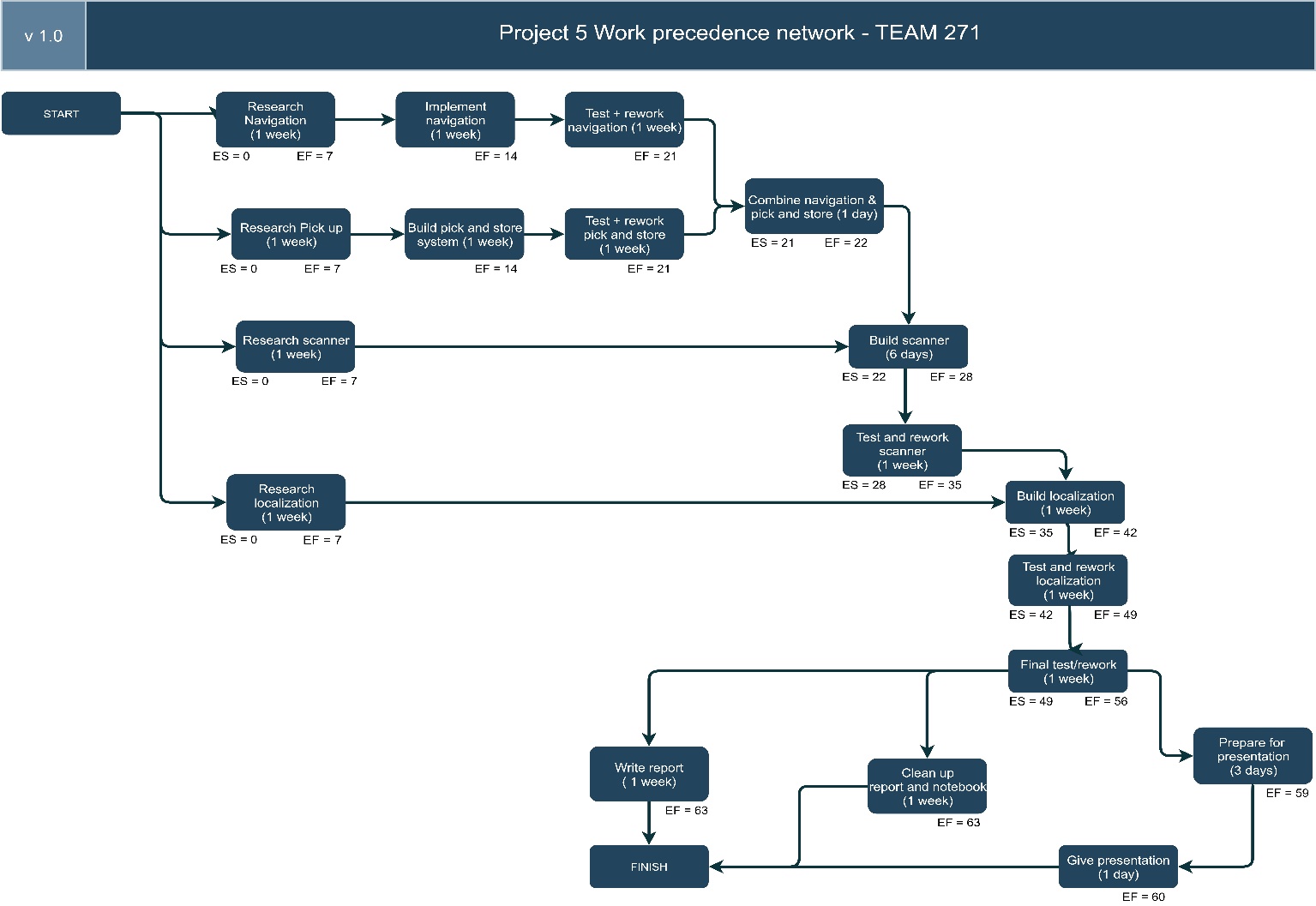


Figure : Project 5’s Work precedence network

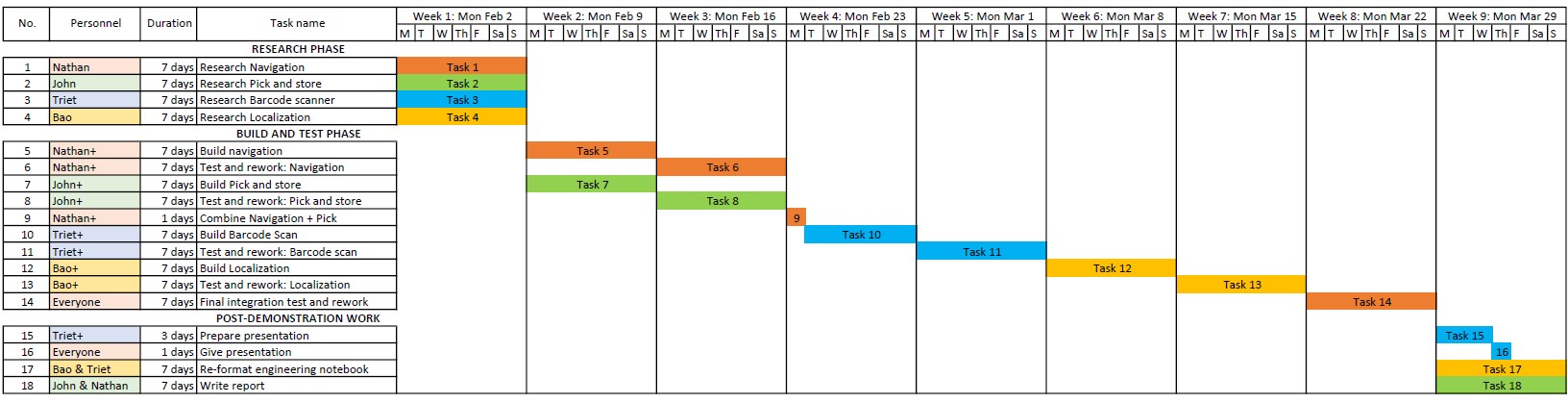


Figure : Project 5's Gantt Chart

# Meeting 2: Navigation system – 02/14/2020

*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:

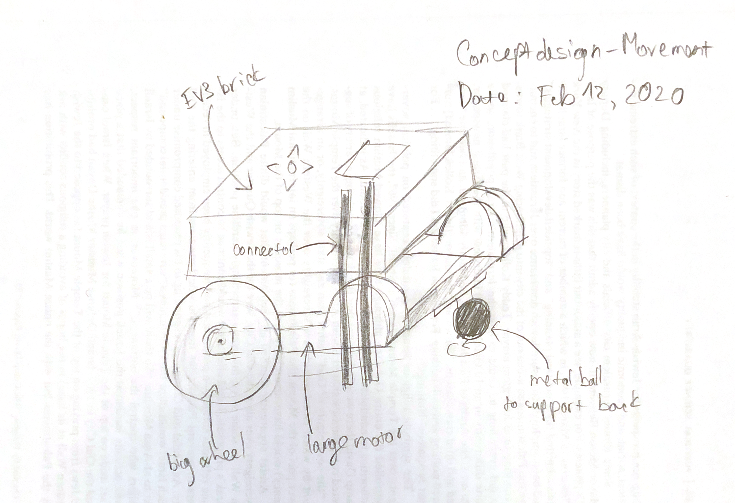


Figure : Conceptual design for robot movement system

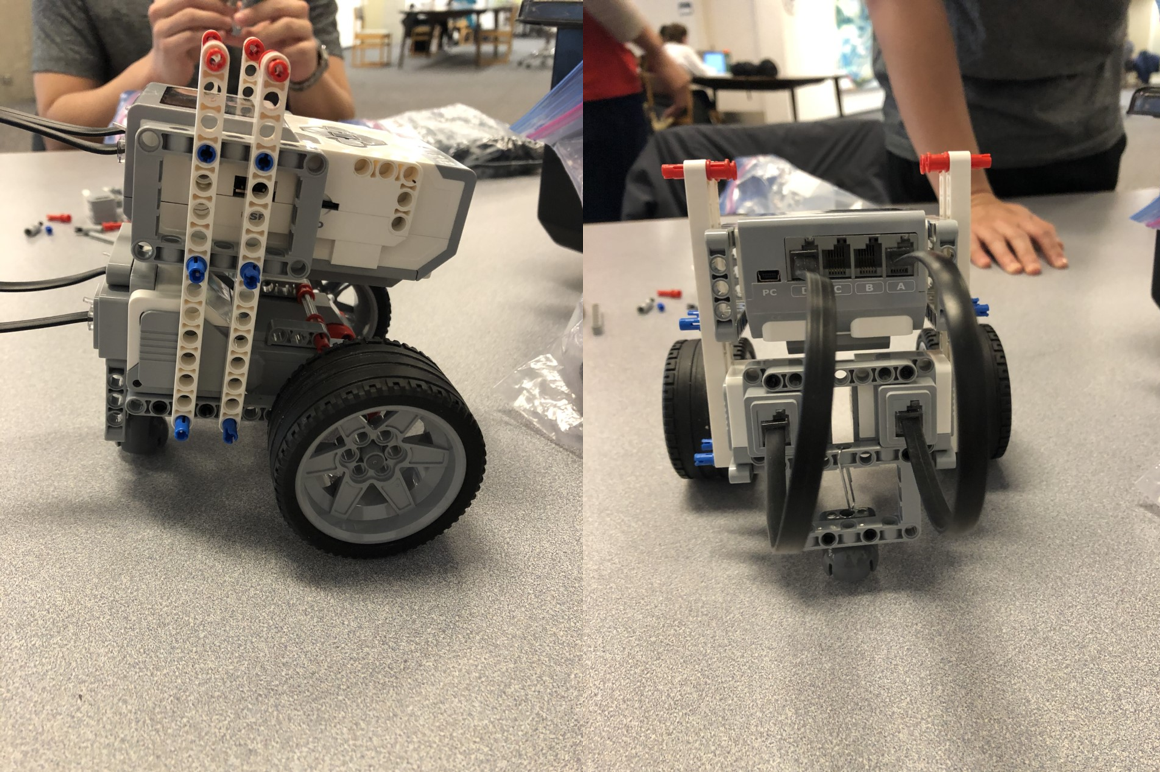


Figure : Prototype of robot movement system

## 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 veer 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:

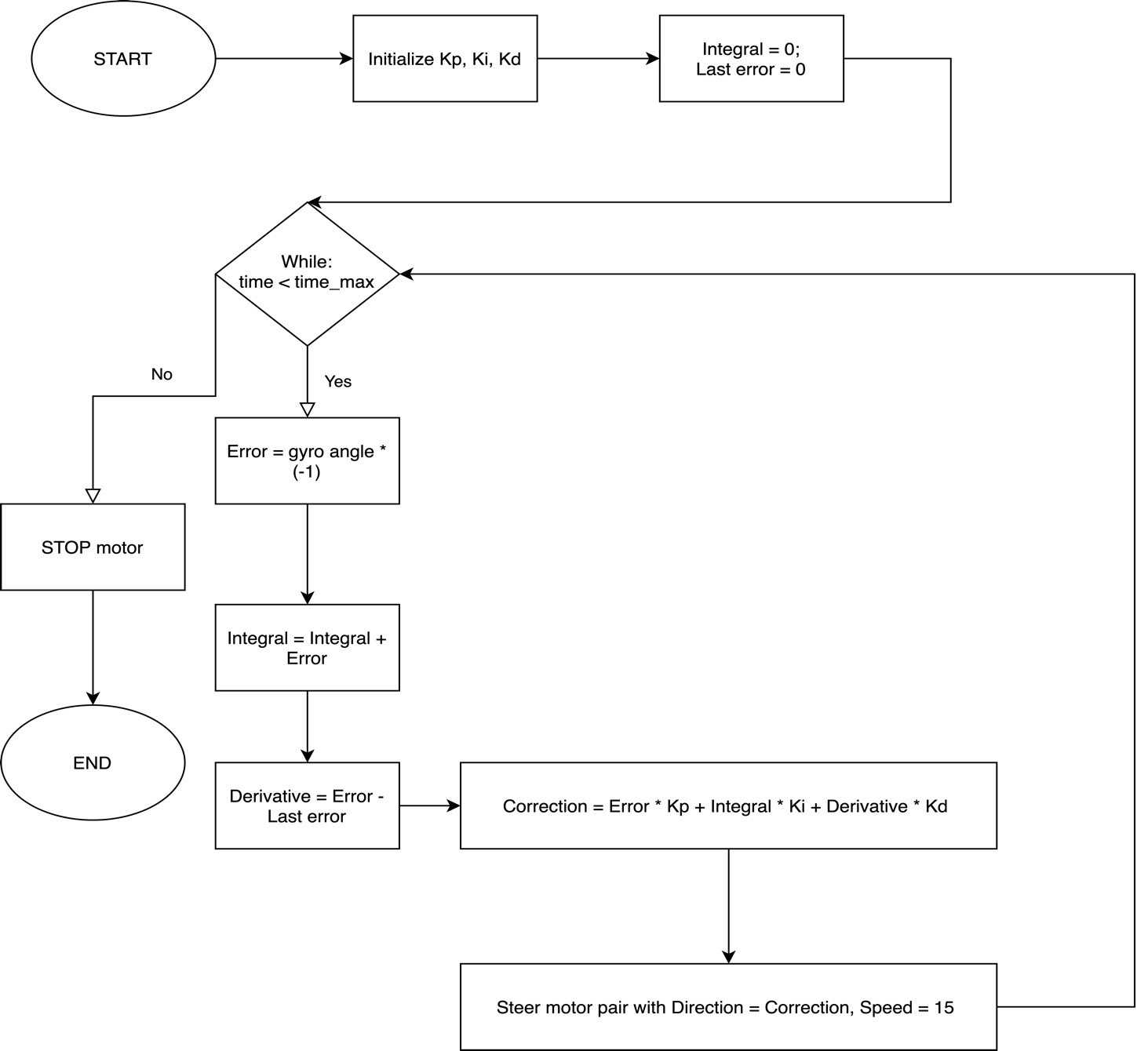


Figure : Flowchart for PID going straight algorithm

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 flowchart:

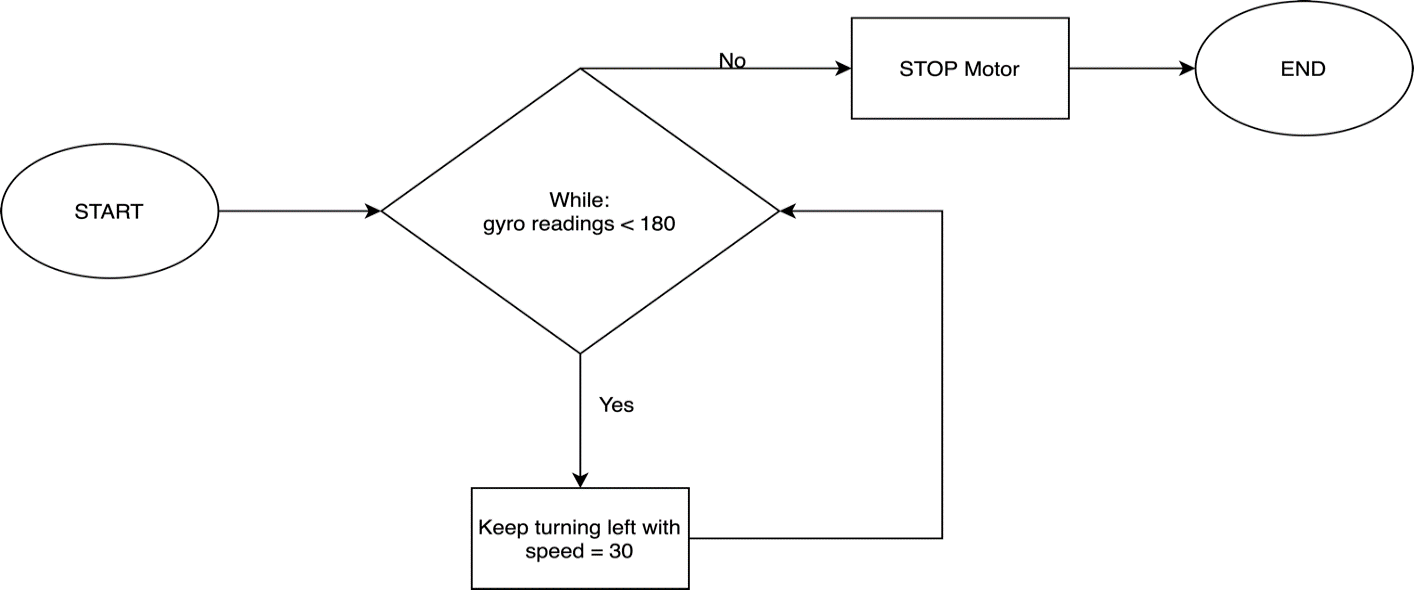


Figure : Flowchart for turning 180 algorithm v.1

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:

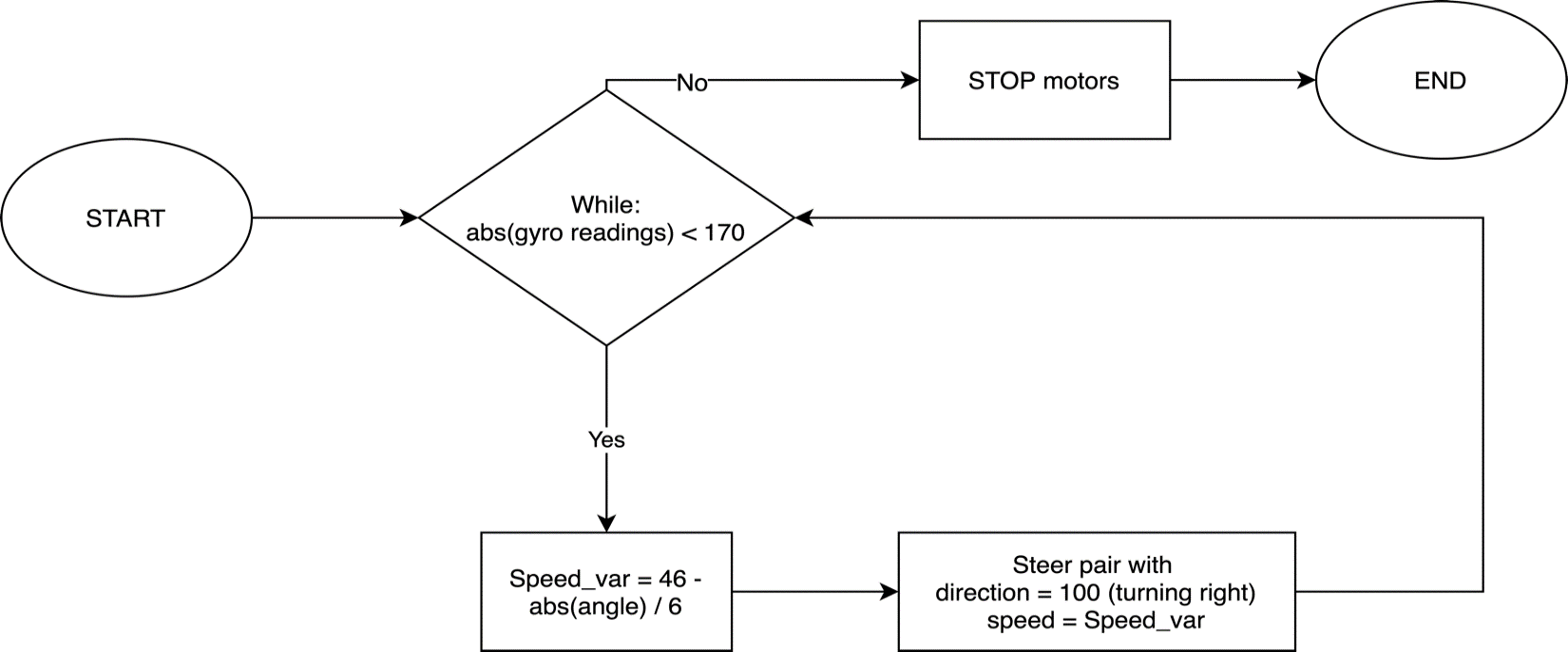


Figure : Flowchart for turning 180 algorithm v.2

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:

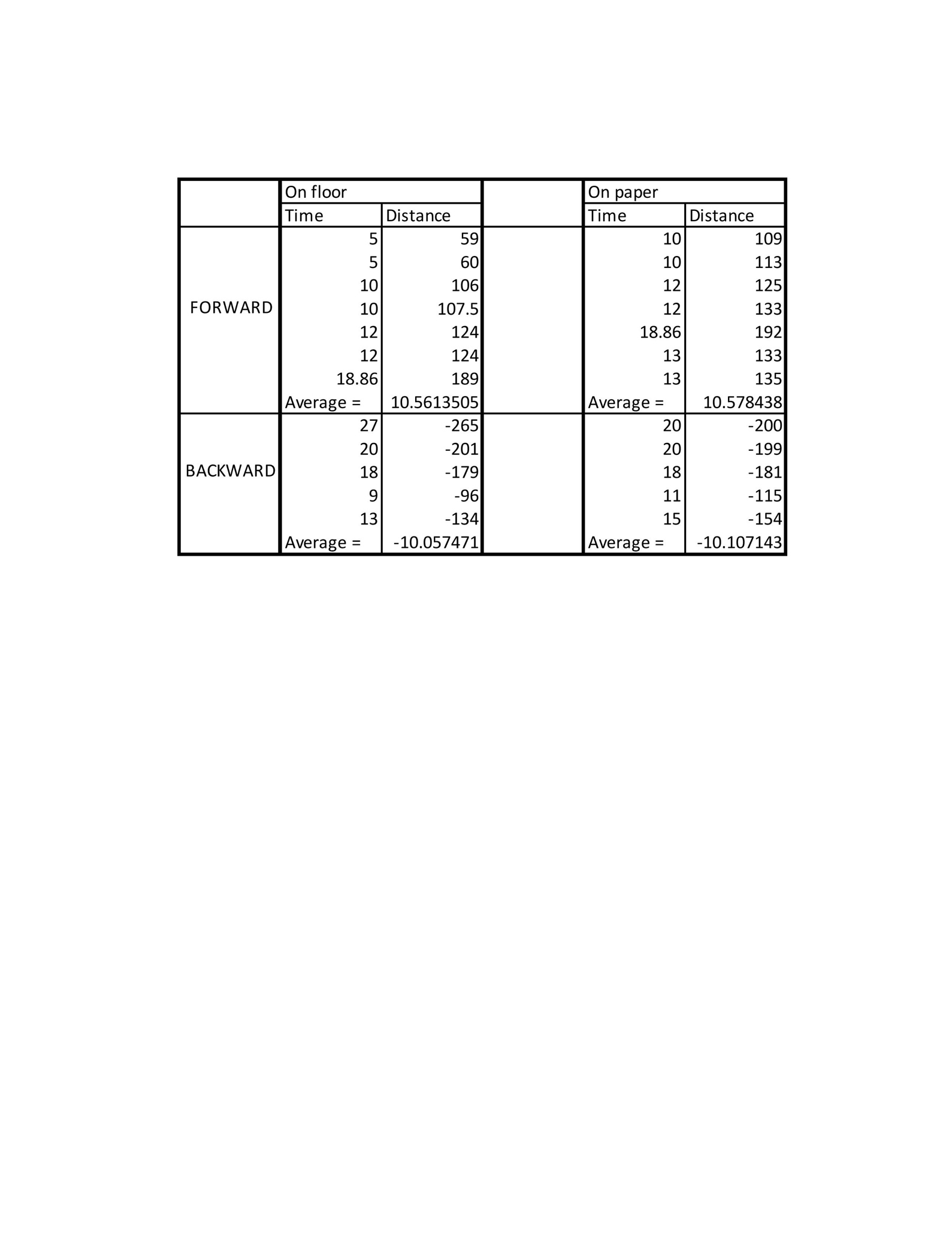
Testing going forward and going backward individually:

Figure : 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.

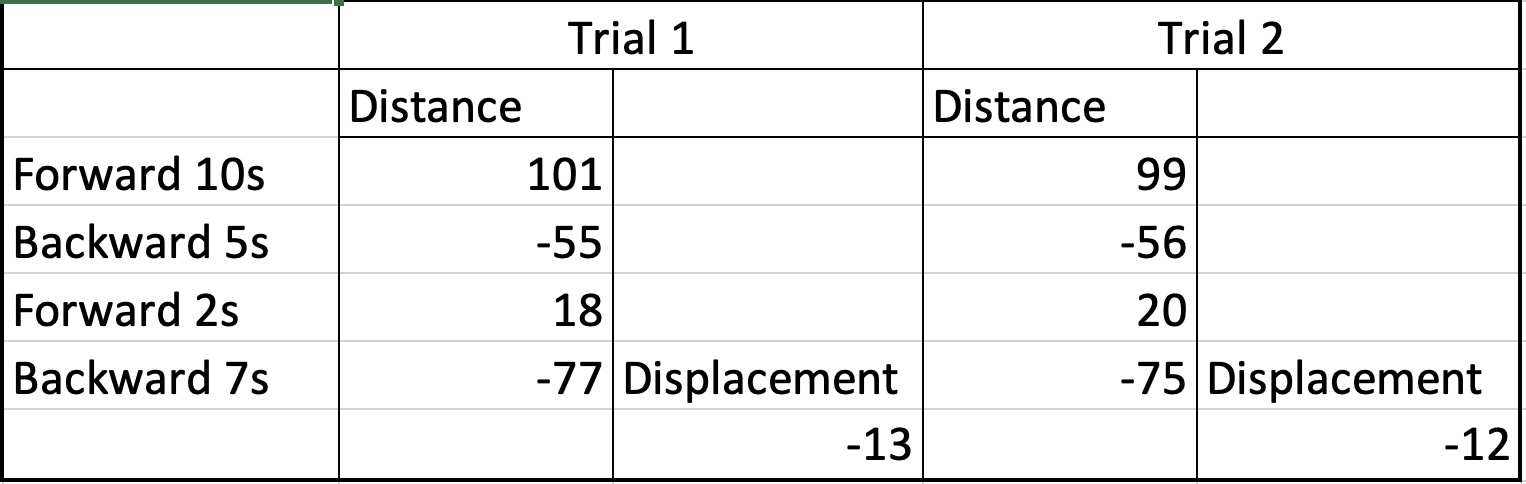
Testing going forward and backward together:

Figure : 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 : 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 – 02/19/2020

*Feb 19, 2020  
Langsam Library, 4pm – 6pm*

***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](#_3._Building_the)) 4. We tested the pickup system using different weights we had ([details below](#_4._Functionality_testing:)) |

**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:

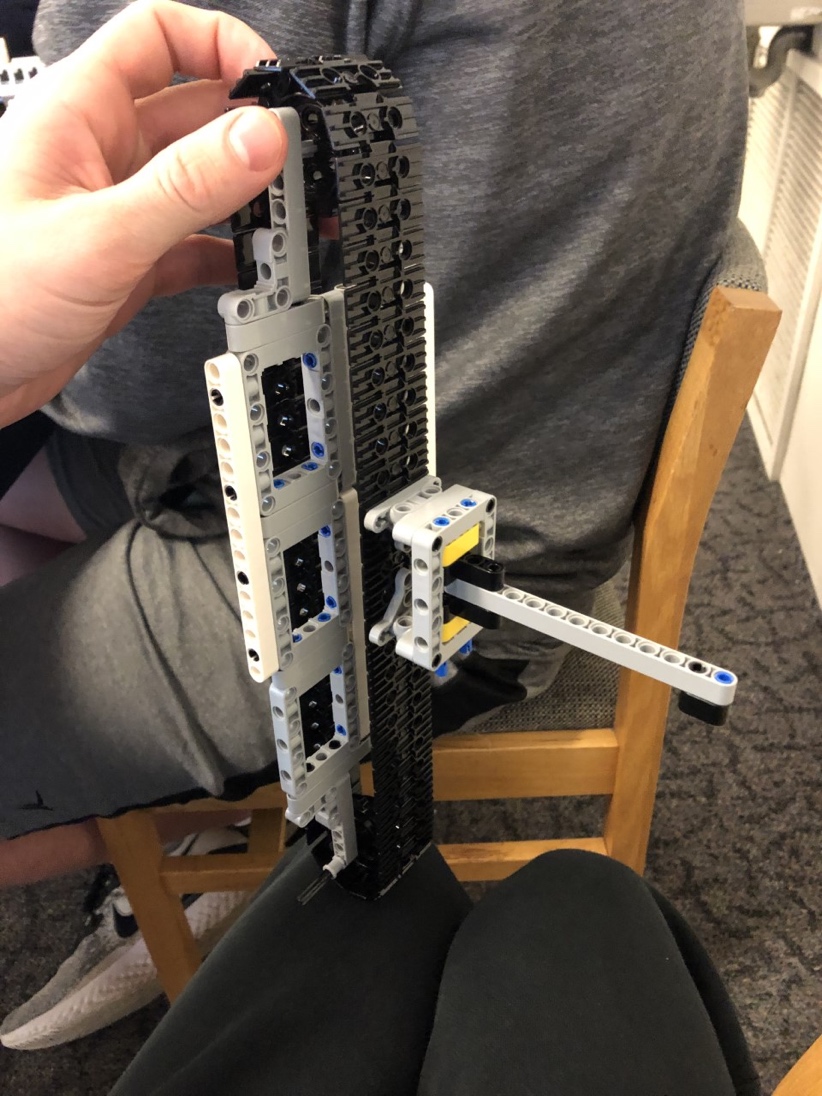


Figure : Conveyor belt frame + pickup arm for the lift system

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:

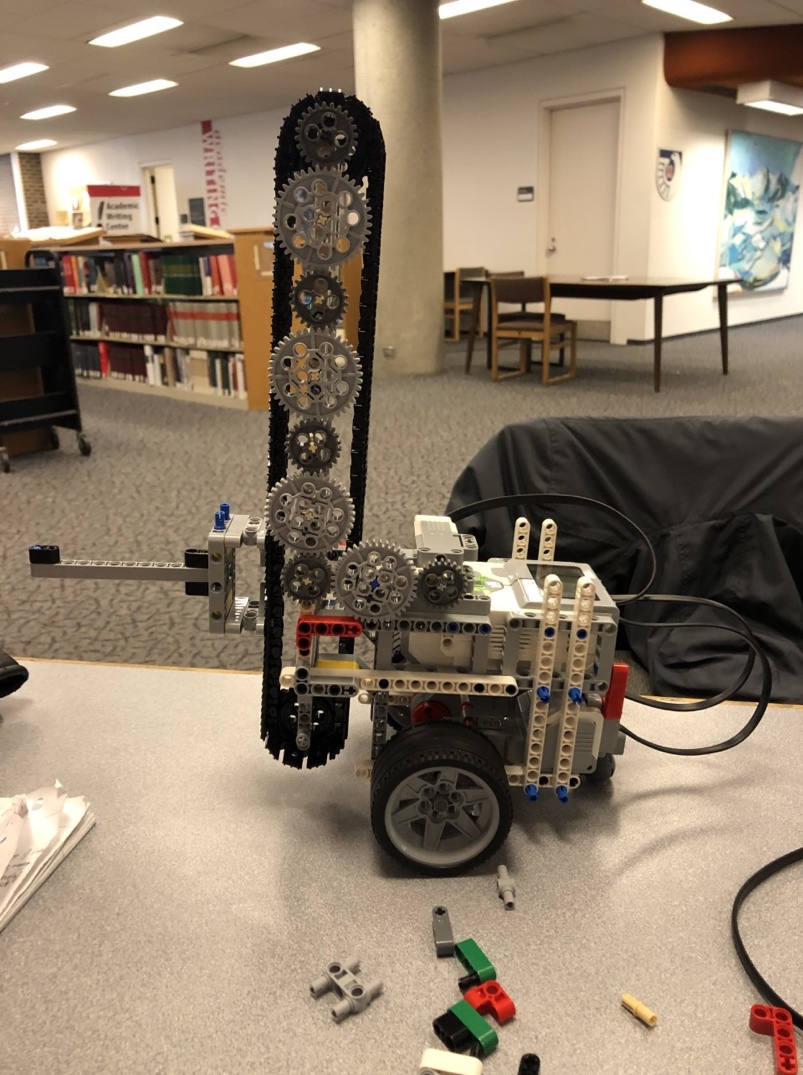


Figure : Final design for robot’s lifting mechanism

## 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 object, which exceeded the weight requirement in Project 5’s Request for Proposal. Therefore, we concluded our design as successful.

# Meeting 4: Ultrasonic sensor and color sensor – 02/26/2020

*February 26, 2020.   
Langsam Library, 4pm – 5:30pm*

***Attendance:*** *Everyone*

|  |  |
| --- | --- |
| Task | Reflection |
| 1. Improve lift system with an ultrasonic sensor 2. Work on barcode scanner system 3. Continue reinforcing the robot with the added weight | 1. We installed an ultrasonic sensor to allow for the robot to know when to stop while approaching the box ([details below](#_1._Ultrasonic_sensor:)) 2. We installed a color sensor to allow the robot to read the barcode on the box ([details below](#_2._Color_sensor)) 3. With added mass from two extra sensors and the box (when picked up), we reinforced the robot with a stopper in the front and more support for the back ([details below](#_3._Extra_reinforcement)) |

**Details:**

## 1. Ultrasonic sensor:

To improve the functionality of our lift system, we added an ultrasonic sensor to the lift system so that the robot knows when it has gotten close enough to the box, then it will pick up the box if the barcode system returns true, and will back off otherwise.

We decided the install the ultrasonic close to the ground because the sensor has to continuously look for boxes and obstacles. We intend the design an algorithm for the robot to move only when the ultrasonic sensor is not blocked. If it is blocked means that a box is front, the robot will trigger the pickup system. Then, after the box has been lifted, the ultrasonic is no longer blocked, the robot will move again.

## 2. Color sensor and barcode scanner system:

We also started working on the barcode scanning system by install a color sensor for the robot. Placement of the color sensor had to be precisely at 6 inches from the ground because that is also the height of the box where the barcodes are attached, and we have no way to move the color sensor around. To scan the barcodes, we discussed that, since we have no more motors to move the color sensor, we will let the robot turn gradually, and read each color section individually (there are 4 black-white sections that make up the barcodes). The reading results can then be mapped into the corresponding type of box in the Python code.

A close up of text on a white background

Description automatically generated

Figure : Concept design for sensors layout

## 3. Extra reinforcement for added weight:

With added weight from two extra sensors and the box, we reinforced the robot with a stopper in the front and more support for the back and middle parts.

# Meeting 5: Going online and completing subtask 2 – 04/17/2020

*April 17, 2020  
Facetime Virtual meeting, 7pm – 8:30pm*

***Attendance:*** *Everyone*

|  |  |
| --- | --- |
| Task | Reflection |
| 1. Discuss new project requirements including subtask 2, and decide on options for team online meeting during social distancing order 2. Brainstorm ideas for simulated navigation system 3. Brainstorm ideas for simulated barcode scanner system | 1. We redefined our objectives for project 5, including subtask 2 and agreed to make Facetime weekly virtual meeting during Covid-19 pandemic. Bao was assigned the main programming responsibility. ([details below](#_1._Project_5)) 2. We constructed an algorithm allowing the robot to navigate through shelves and across quads in the arena to check boxes ([details below](#_2._Navigation_system)) 3. We constructed a back-track scanning algorithm to allow the robot to stop and scan barcode when robot encounters a box ([details below](#_3._Barcode_scanning_1)) |

**Details:**

## 1. Project 5 going online and subtask 2:

As the Covid-19 virus pandemic broke out and face-to-face interactions are suspended, our team cannot get together to build and discuss the LEGO robot anymore. Therefore, a new set of requirements for completing Project 5 in an online manner was published. Our team identified the main points:

* Conceptual designs will now replace concrete prototypical robot construction.
* Simulation code will now replace EV3 brick code. The simulation will comprise of 4 components: Navigation system (Robot is able to move around the arena), Barcode scanner system (Robot has barcode scanning actions and is able to identify the box with the correct barcode), Collision avoidance system (Robot is able to maneuver around obstacles on the field), and IPS system (Robot is able to tell its position in the field along with radial distances from HOME A, C, D)

The next milestone in Project 5 is Subtask 2, which now requires us to film videos summarizing our progress on the physical and logical components of the robot. Because our team have fully constructed the robot’s physical parts prior to the social distancing order, with appropriate images and concept designs, we believe we have the physical requirement satisfied. We will need to work on the simulation part, starting with thinking of algorithms for scanning barcode, moving around the arena, and knowing exactly where the robot should go next.

As our team will come together to think of language-agnostic algorithms, Bao will be assigned the main coding responsibility because Bao has had experiences creating Python simulation before.

A person looking at the camera

Description automatically generated

Figure : Image of team 271 videoconferencing

## 2. Navigation algorithm:

Because the arena is represented as an Oxy plane with Ox → and Oy ↑, we constructed a simple navigation algorithm that allows the robot to move from current point to new point as followed:

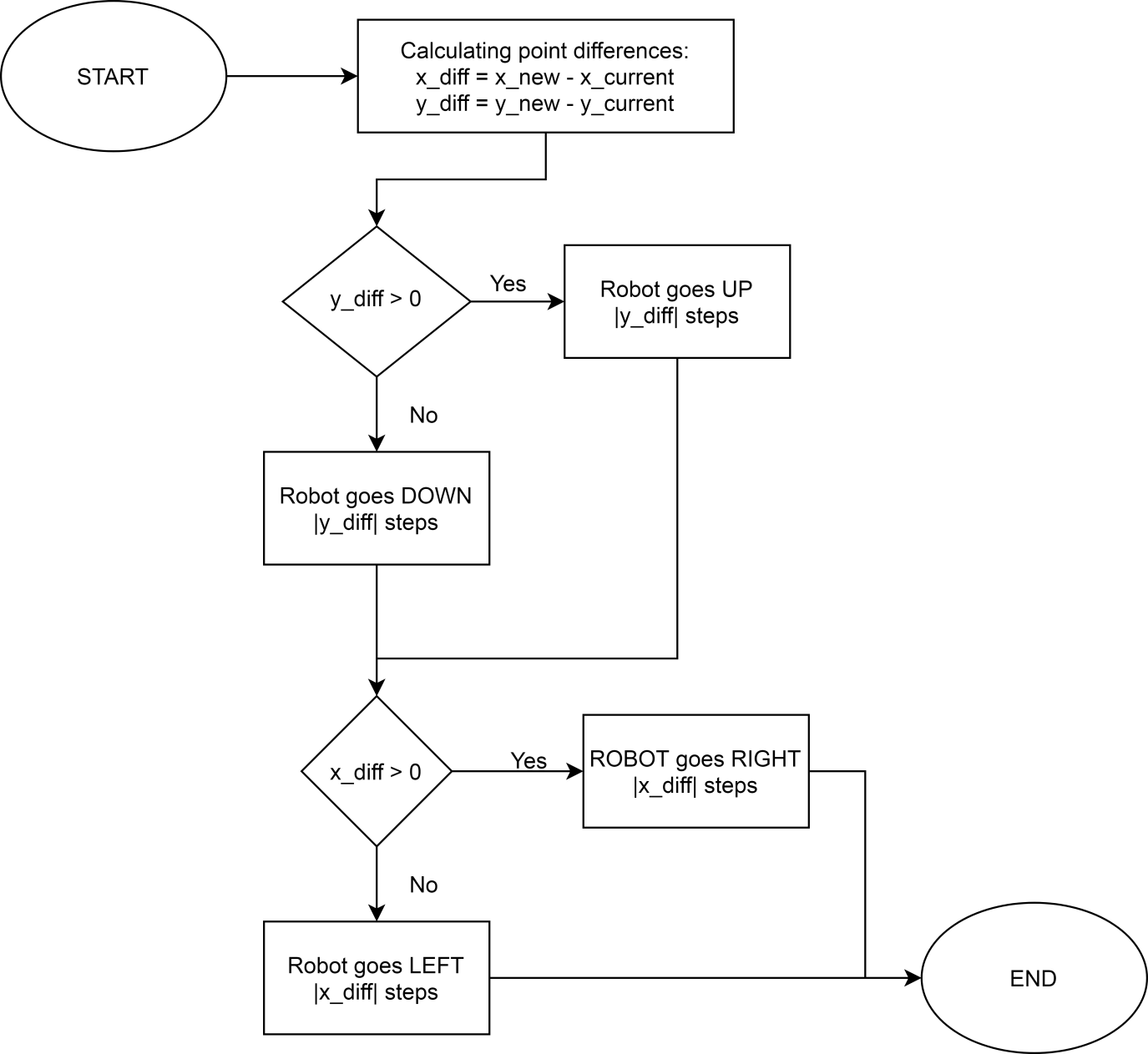


Figure : Flowchart for algorithm to navigate between 2 points

The navigation between two points algorithm starts with calculating the dimensional differences between the two point: the difference in x coordinates and the difference in y coordinates. Then, based on the signs of the differences, the robot decide which direction it needs to go; specifically:

* x\_diff > 0 means that the new point has bigger x coordinates, and therefore is on the right of the robot’s current position. Therefore, robot goes RIGHT |x\_diff| steps. Similarly, if x\_diff < 0, robot goes LEFT
* y\_diff > 0 means that the new point has bigger coordinates, and therefore is above the robot’s current position. Therefore, robot goes UP |y\_diff| steps. Similarly, if y\_diff < 0, robot goes DOWN

For example, if robot is currently at (54, 101) and needs to go to point (55, 7), robot will first to DOWN 94 steps to get to (54, 7), and then go RIGHT 1 step to go to (55, 7)

One problem we detected with this algorithm is robot will always navigate on the y-direction first then the x-direction, so this navigation algorithm only works if invoked when robot is in vertical hallways (where there should be no obstacles blocking the y-direction). If the algorithm is invoked if the robot is at the middle of a shelf, the robot may traverse over the boxes. Therefore, we fix this problem by incorporating it into a broader navigation sequence where it will only be used only when the robot is at the hallways. Our solution is more clearly described in the flowchart followed:

A close up of text on a white background

Description automatically generated

Figure : Flowchart for algorithm to navigate entire arena

In this algorithm to navigate the entire area, we make sure the robot has entered the hallway (after finished searching a shelf horizontally) before invoking the 2-point-navigation algorithm (indicated by **“navigate”** in bold), thus avoiding the aforementioned issue.

In the flowchart above, we also used a smaller algorithm (signaled by (\*\*) symbol) to determine where to go next from the robot’s current position. The flowchart for this algorithm is as followed:

A close up of a map

Description automatically generated

Figure : Flowchart for algorithm to determine where to go next

In this “Where to go next” algorithm, if the robot is at a “quad-terminal position” (that is, the position the robot will be after it finished searching through all 4 shelf lines in a quad), then the next position it needs to go is the beginning position of the next quad. Else, if robot has not finished all 4 shelf lines in quad, then the next position is the beginning of the next shelf of the current quad. Numbers like 7, 31, 77, 101, 22, or 2 are numbers that our team calculated in order to implement this algorithm, by leveraging some mathematical facts of our simulation that: When the robot ultrasonic sensor touched a box, its center is 5 inches away from the shelf border, or that the hallway between the shelf are all 12 inches, etc.

## 3. Barcode scanning algorithm:

To scan the barcode using the color sensor, our team saw that the color sensor can only read 1 color bit at a time, meaning after reading the first color bit, and save the result into a temporary list, the robot would need to “nudge” slightly so that the color sensor aligns with the next color bit, then repeat reading, storing, and “nudging” until all 4 color bits have been recorded. To do this, the robot must first be situated at the beginning of the box and begin color-bit-reading 4 times while moving in the scanning direction.

However, due to the randomness of the location of the box, it is impossible to hardcode the beginning positions of the box for the robot to arrive there precisely. Therefore, our team implement a backtrack scanning algorithm: When the ultrasonic sensor detects a box, the robot then moves backward 1 step, then check again to see if the ultrasonic sensor still detects the box, and repeat backward-then-check sequence until the box is not seen anymore, which means the robot is now at the beginning at the box. The algorithm is more clearly described in the illustration and flowchart below:

A close up of text on a whiteboard

Description automatically generated

Figure : Illustration for backtrack-scanning algorithm

A close up of a map

Description automatically generated

Figure : Flowchart for backtrack-scanning algorithm

As for how this algorithm will be used in the bigger picture, we decided that, since each box is 4 inches wide, when the robot is scanning a shelf, it will **stop every 4 inches (4 steps) to turn into the shelf and check if any box is there**. If yes, it will activate the backtrack scanning sequence. If no, it will continue looping again, going 4 inches, and stop to check, until its coordinate has reached the end of the shelf, which will be 36 inches away from the start of the shelf. A special case should be noted is that, if the robot found the correct box, it will pick up the box, break out of the loop, and go straight out to the hallway in order to get back to the home base.

# Meeting 6: Coding the simulation and finalize subtask 2 – 04/19/2020

*April 19, 2020  
Facetime virtual meeting, 8pm – 8:30pm*

***Attendance:*** *Everyone*

|  |  |
| --- | --- |
| Task | Reflection |
| 1. Complete our code for the simulation 2. Film videos for subtask 2 | 1. We completed our simulation code in Python, which includes visual representation of the robot and boxes, and navigation + barcode scan algorithm. ([details below](#_1._Simulation_code:)) 2. We filmed 2 videos on our physical and logical progress and submitted them on BlackBoard ([details below](#_2._Submitting_videos)) |

**Details:**

## 1. Simulation code:

From the simulation started guide from BlackBoard, Bao wrote a simulation program in Python which included:

* Visual representation:
  + The robot is illustrated as an orange 4x6 rectangle
  + Robot’s ultrasonic vision is illustrated as a red 1x2 rectangle in front of the robot’s head.
  + Robot’s color sensor is illustrated as a blue 1x1 square on the left side of the robot.
  + The boxes are illustrated as 4x4 squares, with gray boxes indicating wrong boxes and red boxes indicating boxes with correct barcodes.
  + The home bases are illustrated as purple circles on 4 corners of the field.

A screenshot of a cell phone

Description automatically generated

Figure : Visual representation of team 271's simulation

* Logical codes:
  + The code for our back-track scanning algorithm is as followed:

def backtrack\_scanning(self):

"""

Backtrack scanning algorithm when robot has met a box

"""

# After colliding with the box, the robot will step backward 1 step at a time

# while also turning to check if the box is still there.

# After some back steps, the box is not detected any more, meaning the robot

# has reached the leftmost edge of the box.

# Then it begins scanning in the scanning direction

# Backward until box is no longer seen

while self.robot.ultrasonic\_detection(self.backend.board):

self.robot\_turn\_right()

self.robot\_backward(1)

self.robot\_turn\_left()

plt.pause(0.02)

# When the box is no longer seen, the ultrasonic field is at the box's

# leftmost edge. Now turn right, preparing to forward to scanning position

self.robot\_turn\_right()

# Visual Bug: If robot’s scanning direction is RIGHT, needs to forward 4 steps for

# the colorsensor to align with the first barcode bit

# If robot is scanning in LEFT direction, only needs 3 steps

self.robot\_forward(4 if self.robot.direction == RIGHT else 3)

self.robot\_turn\_left()

plt.pause(0.05)

# Start scanning the 4 bits of the bardcode

full\_code = []

for \_ in range(4):

temp\_bit = self.robot.scan\_barcode(self.backend.board)

if temp\_bit > 0:

full\_code.append(temp\_bit)

else:

print("Code scanning weird")

self.robot\_turn\_right()

self.robot\_forward(1)

self.robot\_turn\_left()

plt.pause(0.007)

# Visual bug: If direction is DOWN (corresponding to LEFT scanning direction)

# must go 1 more step (reason explained in another "Visual Bug" comment above)

if self.robot.direction == DOWN:

self.robot\_turn\_right()

self.robot\_forward(1)

self.robot\_turn\_left()

return full\_code

* + The code for our algorithm to search a shelf horizontally is as followed:

def search\_shelf(self):

"""

In this sequence, the robot scans a whole shelf line to search the right box,

turning inside every 4 inches to check for box.

Go into hallway after finished searching the shelf.

Invoke this sequence only when robot has been near a shelf,

facing towards the boxes

"""

scanning\_direction = self.get\_scanning\_direction()

x\_begin = self.robot.center[0]

x\_end = x\_begin + 38 \* scanning\_direction[0]

print(f"Scanning seq start with {x\_begin}, {x\_end}")

while min(x\_begin, x\_end) <= self.robot.head[0] <= max(x\_begin, x\_end):

# Only continue searching if has not found the right box (storage is empty)

if self.robot.storage\_empty:

box\_detected = self.robot.ultrasonic\_detection(self.backend.board)

if box\_detected:

full\_code = self.backtrack\_scanning()

plt.pause(0.007)

# If barcode is correct, proceed to get into position to store box

if tuple(full\_code) == self.target\_code:

# Go a bit backward to place the robot right in middle of box

self.robot\_turn\_right()

self.robot\_backward(3)

self.robot\_turn\_left()

# Get close to the box

self.robot\_forward(1)

# Pick up the box and move backward

self.robot\_pick\_box()

self.robot\_backward(1)

self.robot\_turn\_right()

else:

self.robot\_turn\_right()

# Forward 4 inches then turn back in to see if any boxes appear

self.robot\_forward(4)

self.robot\_turn\_left()

# If a box is found and picked up already, just go:

else:

self.robot\_forward(1)

print(f"Scanning seq ended. Robot currently at {self.robot.center}")

# Go to hallway by moving 2 extra steps from the shelf’s end

self.robot\_goto\_point(add\_tuple(tuple((x\_end, self.robot.center[1])),

mult\_tuple(scanning\_direction, 2)))

print(f"Went into hallway at {self.robot.center}")

* + The code for our navigation algorithm is as followed:

def search\_entire\_area(self):

"""

Combine other sequences into a whole box search sequence on entire field

"""

game\_finished = False

num\_quad\_finished = 0

starting\_position = self.robot.center

while not game\_finished:

# Depart from home

self.escape\_home()

plt.pause(0.5)

# Make first search

self.search\_shelf()

# Continuously do subsequent searches if necessary

while self.robot.storage\_empty and num\_quad\_finished < 4:

next\_point, next\_scan\_direction = self.where\_to\_go\_next()

print(next\_point, next\_scan\_direction)

# Move into next spot

self.robot\_goto\_point(next\_point)

self.robot\_become\_direction(COUNTER\_CLOCKWISE[next\_scan\_direction])

# Scan a shelf line

self.search\_shelf()

print(f"Current quad is: {self.robot.quad}")

if self.robot.center in QUAD\_END:

num\_quad\_finished += 1

print(f"Finished quad {self.robot.quad}")

# If carrying a box, initiate go\_home sequence

if not self.robot.storage\_empty:

self.go\_home(starting\_position)

game\_finished = True

else: # Meaning no box found

print("Searched 4 quads but no boxes found")

game\_finished = True

## 2. Submitting videos for subtask 2:

We filmed 2 videos on our physical and logical progress and submitted on BlackBoard. Our physical component was 100% completed as we have fully built the robot with all the sensors before the social distancing order. Our logical progress only included progress on the Navigation system and Barcode scanning system; we have yet to complete the Collision Avoidance algorithm and Indoor Positioning system (IPS).

# Meeting 7: Collision Avoidance system algorithm – 04/23/2020

*April 23, 2020*

*Facetime virtual meeting, 7pm – 8pm*

***Attendance:*** *Everyone*

|  |  |
| --- | --- |
| Task | Reflection |
| 1. Refine barcode scan algorithm 2. Develop Collision avoidance system 3. Code the collision avoidance algorithm | 1. We realized our barcode scanner algorithm did not work well when the robot faced 2 adjacent boxes. So we refined our algorithm to solve this issue ([details below](adjacent#_1._The_)) 2. We brainstormed a collision avoidance algorithm to circumvent stationary obstacles ([details below](#_2._Navigation_system)) 3. We added the collision avoidance code into our simulation ([details below](#_3._Barcode_scanning_1)) |

**Details:**

## 1. The “adjacent boxes” problem:

After running our simulation many times, we saw a consistent problem with our barcode scanning system when there are two boxes next to each other. If box 1 is adjacent to box 2, when the robot encounters the second box, it will "backtrack" until it no longer sees boxes; however, because the two boxes are adjacent to each other, the robot will see them as one long box, and will backtrack to the start of box 1, which is not something our team wants (we just want to the robot to go backward into the starting position of box 2 so we can initiate the color sensor sequence). The following is a drawn illustration of our “adjacent boxes” issue:

A close up of text on a whiteboard

Description automatically generated

Figure : Illustration of the "adjacent boxes" problem

After much discussion, we decided that this is an edge case that we should hardcode into our robot. Triet came up with the idea that we should track the number of backward steps the robot has taken in this backtracking sequence. If the number of backward steps is higher than 4, it means that there must be two adjacent boxes (because one box alone is only 4 inches wide; the robot should not take more than 4 steps to get to the beginning position of the box if that one box is isolated). Then we will adjust the robot accordingly: If the robot “felt” that it has encountered two adjacent boxes, after it meets box 2 and backtracks to the beginning of box 1, it will initiate a **“forward-scanning sequence”**, which is opposite of backtrack scanning – the robot will now go forward until it longer sees box, and at that point, it will be at the rightmost edge of box 2. From there, it will go backward about 4 steps in order to align the color sensor with the first color bit of box 2. The refined algorithm is described in the flowchart followed:

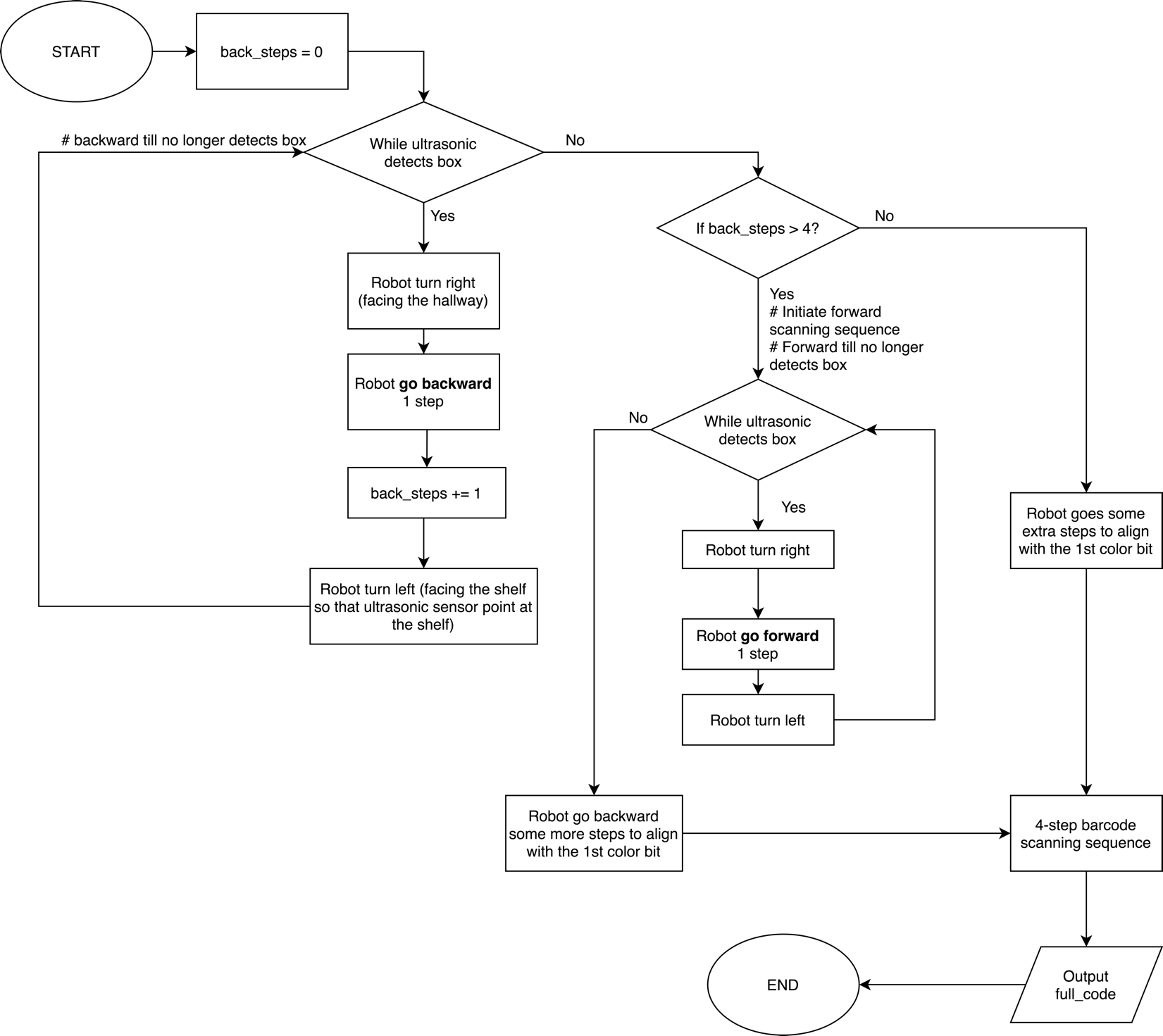


Figure : Flowchart for backtrack scanning algorithm v.2

## 2. Collision Avoidance system:

Our Collision Avoidance system (abbreviated as CAS from now on) will include an ultrasonic sensor to detect if there are objects nearly, and a color sensor to see if that object is a box or an obstacle (a rock).

# 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