Engineering Notebook

TEAM 271
ENED 1120 – 021
Dr. Cedrick Kwuimy
Project 5: Autonomous Record Retriever
Project date: Feb 2, 2020 to April 24, 2020

Team members:
John Cummings | cummijh
Nathan Damas | damasnj
Bao Huynh | huynhlbg
Triet Pham | phamt7

Goal:
To construct an autonomous that can navigate around the field, scan the box's barcode and pick up the box with the correct given barcode.

Table of Contents

Table of Figures

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) 2. We broke down project 5's robots into 5 sub-components (details below) 3. We wrote repetition test plans for each component and make a specification review file (details below) 4. We created work breakdown structure, work precedence network, and Gantt chart for project 5 (details below)

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:

a. **What is a barcode?**

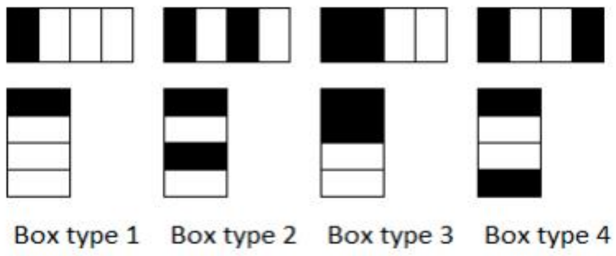


Figure 1. Example of barcodes

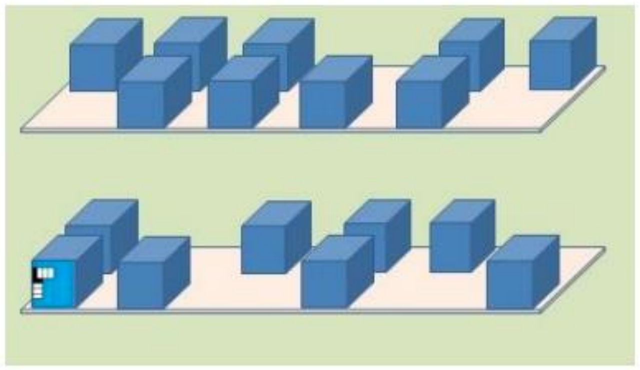


Figure 1: Barcode types and locations

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

b. What will the arena look like?

This is the map of the final arena:

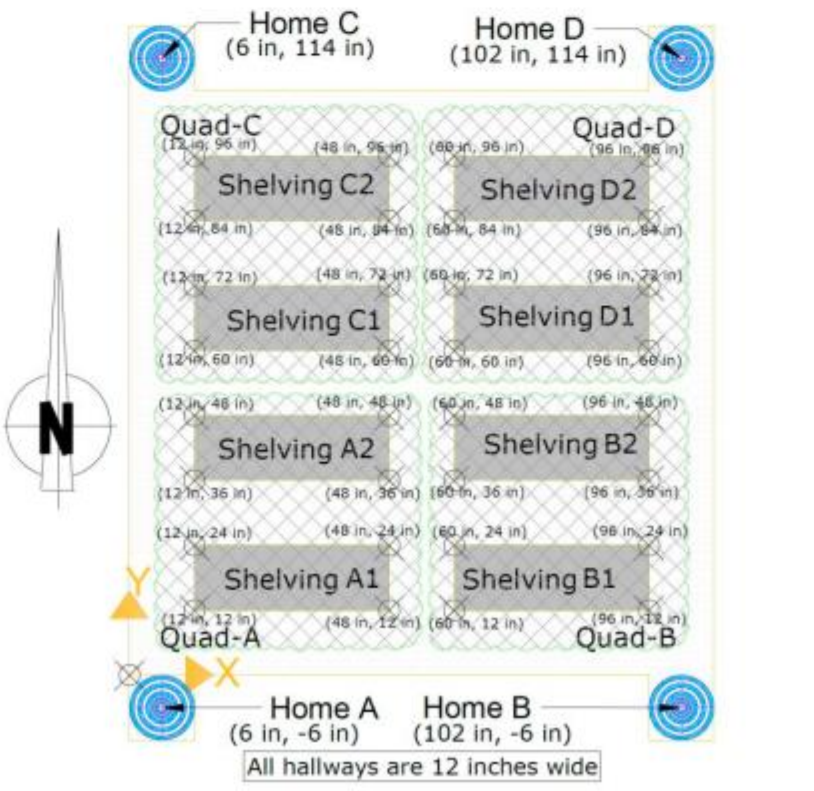


Figure 2: Demonstration arena overview

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).

c. 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.

d. 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.

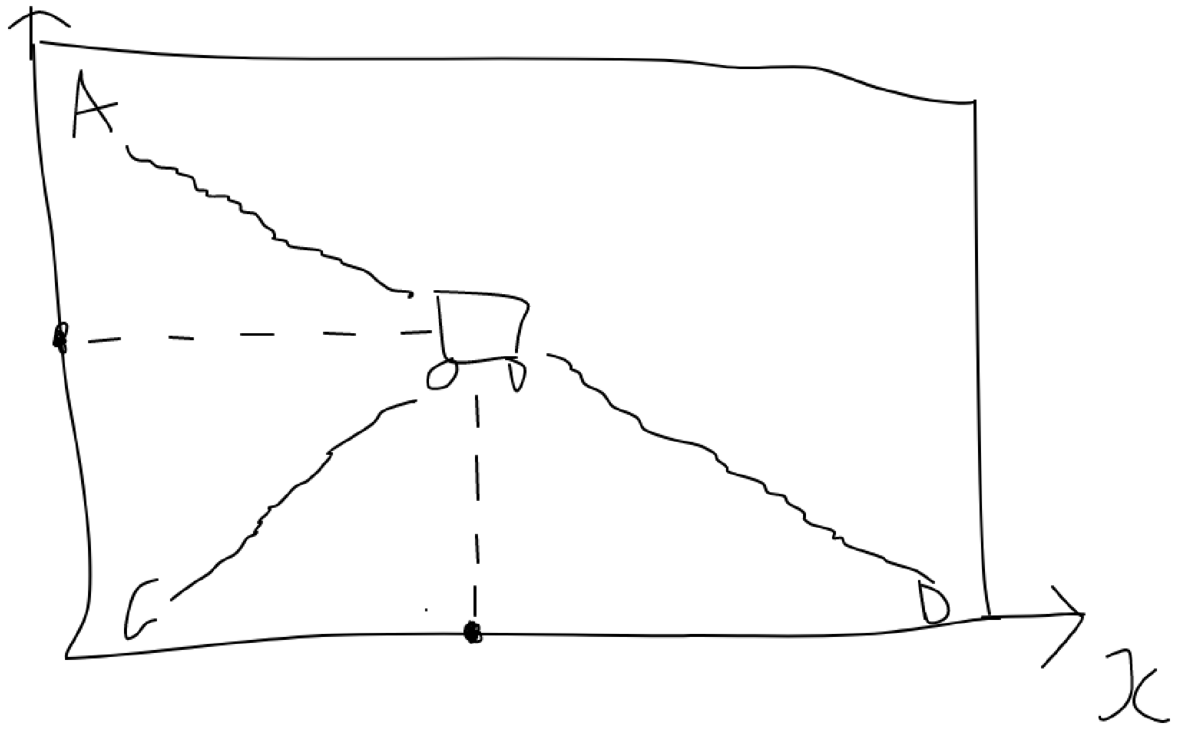


Figure 3: Visualization of IPS System

e. 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

f. 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

g. 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:

A. Navigation system: The robot must first the able to move smoothly in the arena, so a navigation with wheels or treads is necessary
B. Barcode scanning system: The robot must be able to read the barcode correctly to get the right box
C. 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
D. 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
E. 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

A. 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

B. 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

C. 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

D. 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

E. 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:

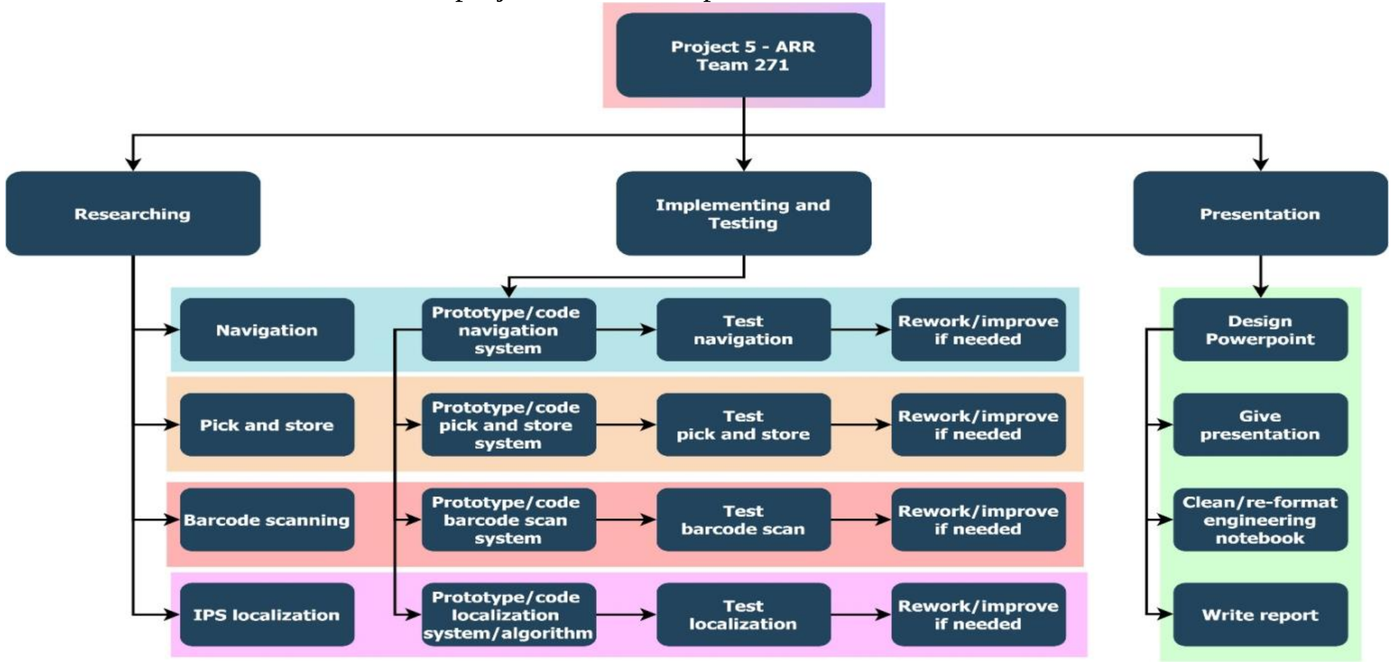
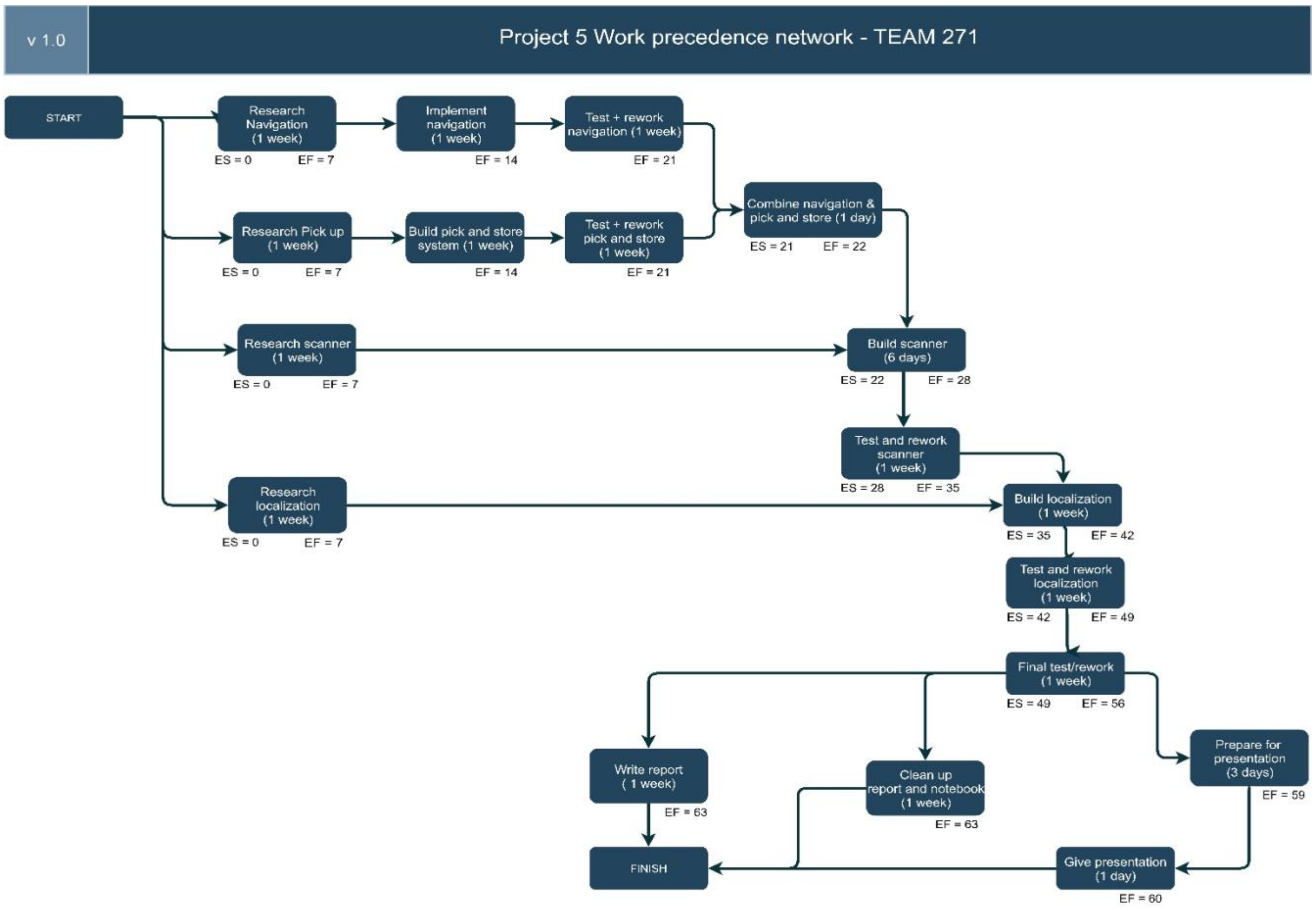
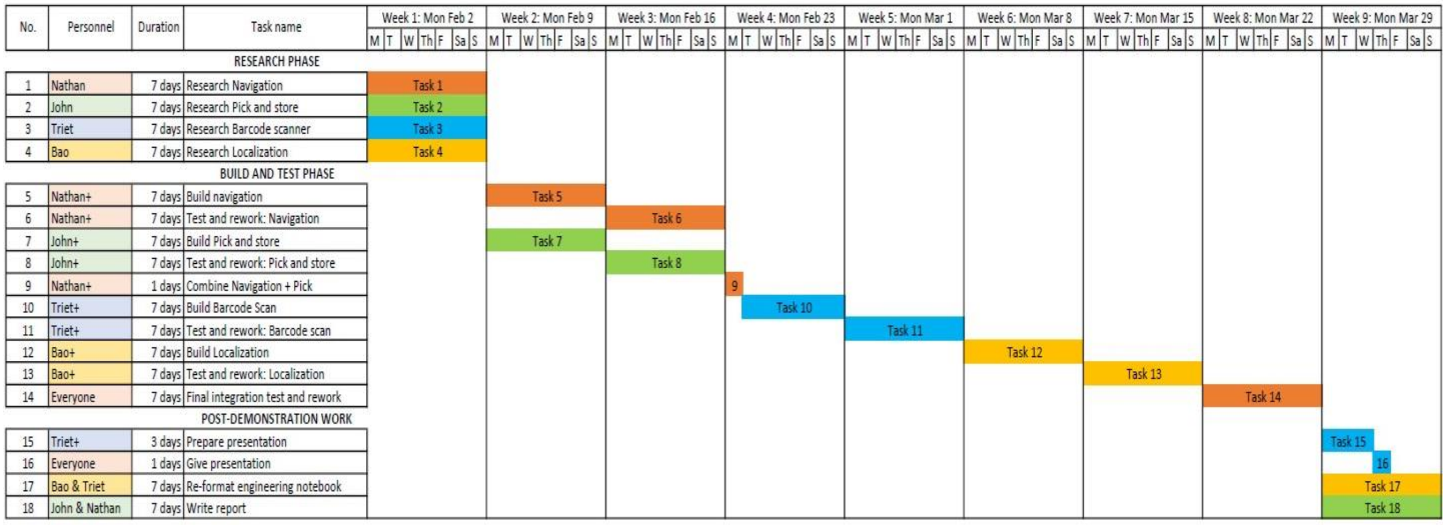


Figure 4: Project 5's work breakdown structure



v 1.0
Project 5 Work precedence network - TEAM 271
START
Research Navigation (1 week)
ES = 0
EF = 7
Implement navigation (1 week)
EF = 14
Test + rework navigation (1 week)
EF = 21
Research Pick up (1 week)
ES = 0
EF = 7
Build pick and store system (1 week)
EF = 14
Test + rework pick and store (1 week)
EF = 21
Combine navigation & pick and store (1 day)
ES = 21
EF = 22
Research scanner (1 week)
ES = 0
EF = 7
Build scanner (6 days)
ES = 22
EF = 28
Test and rework scanner (1 week)
ES = 28
EF = 35
Research localization (1 week)
ES = 0
EF = 7
Build localization (1 week)
ES = 35
EF = 42
Test and rework localization (1 week)
ES = 42
EF = 49
Final test/rework (1 week)
ES = 49
EF = 56
Write report (1 week)
EF = 63
Clean up report and notebook (1 week)
EF = 63
Prepare for presentation (3 days)
EF = 59
FINISH
Give presentation (1 day)
EF = 60

Figure 5: Project 5's Work precedence network



No.	Personnel	Duration	Task name	Week 1: Mon Feb 2	Week 2: Mon Feb 9	Week 3: Mon Feb 16	Week 4: Mon Feb 23	Week 5: Mon Mar 1	Week 6: Mon Mar 8	Week 7: Mon Mar 15	Week 8: Mon Mar 22	Week 9: Mon Mar 29
				M T W Th F Sa S	M T W Th F Sa S	M T W Th F Sa S	M T W Th F Sa S	M T W Th F Sa S	M T W Th F Sa S	M T W Th F Sa S	M T W Th F Sa S	M T W Th F Sa S
			RESEARCH PHASE									
1	Nathan	7 days	Research Navigation	Task 1								
2	John	7 days	Research Pick and store	Task 2								
3	Triet	7 days	Research Barcode scanner	Task 3								
4	Bao	7 days	Research Localization	Task 4								
			BUILD AND TEST PHASE									
5	Nathan+	7 days	Build navigation		Task 5							
6	Nathan+	7 days	Test and rework: Navigation			Task 6						
7	John+	7 days	Build Pick and store		Task 7							
8	John+	7 days	Test and rework: Pick and store			Task 8						
9	Nathan+	1 days	Combine Navigation + Pick				9					
10	Triet+	7 days	Build Barcode Scan				Task 10					
11	Triet+	7 days	Test and rework: Barcode scan					Task 11				
12	Bao+	7 days	Build Localization						Task 12			
13	Bao+	7 days	Test and rework: Localization							Task 13		
14	Everyone	7 days	Final integration test and rework								Task 14	
			POST-DEMONSTRATION WORK									
15	Triet+	3 days	Prepare presentation									Task 15
16	Everyone	1 days	Give presentation									16
17	Bao & Triet	7 days	Re-format engineering notebook									Task 17
18	John & Nathan	7 days	Write report									Task 18

Figure 6: 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) 2. Our Python code implemented a PID algorithm with gyro sensor to help redirect the robot if it veers (details below) 3. Using gyro sensor readings, we programmed the robot to turn until a desired angle is reached (details below) 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)

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:

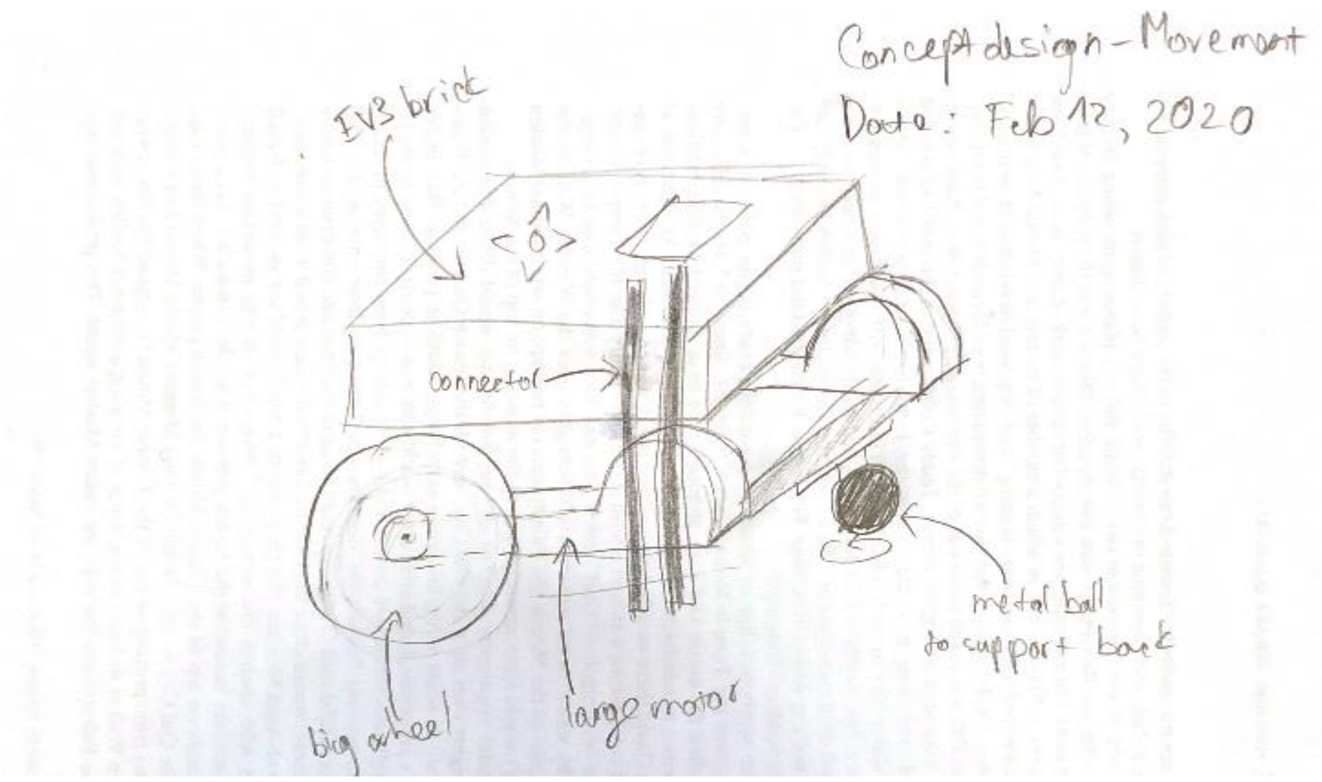


Figure 7: Conceptual design for robot movement system

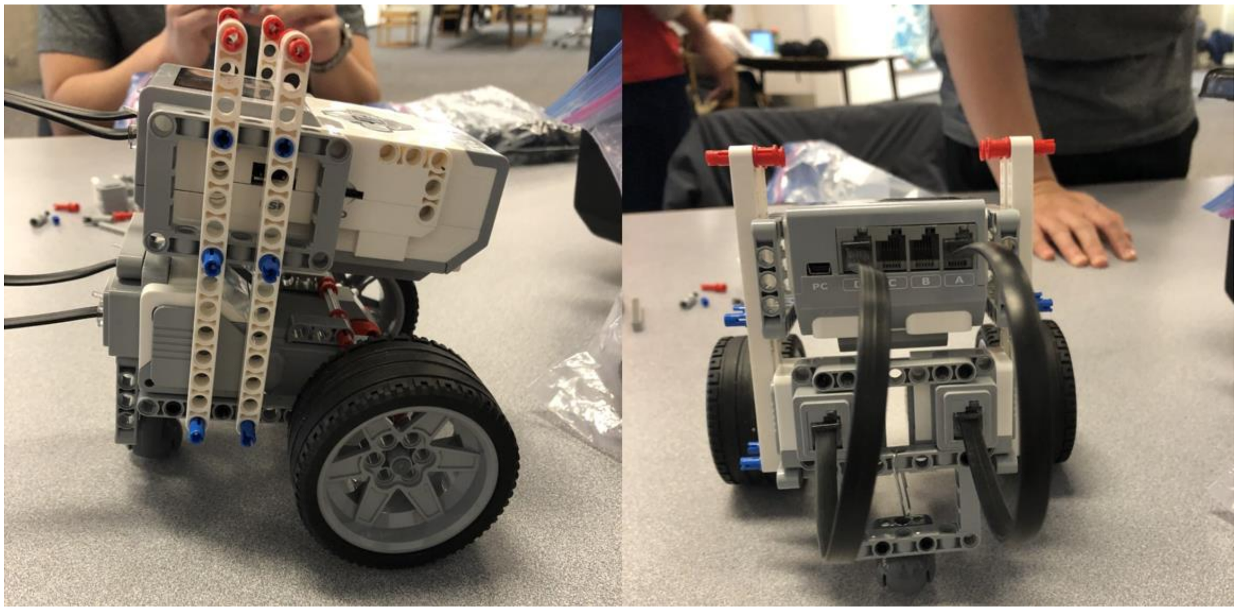


Figure 8: Prototype of robot movement system

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 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), 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), our program is described in the following flowchart:

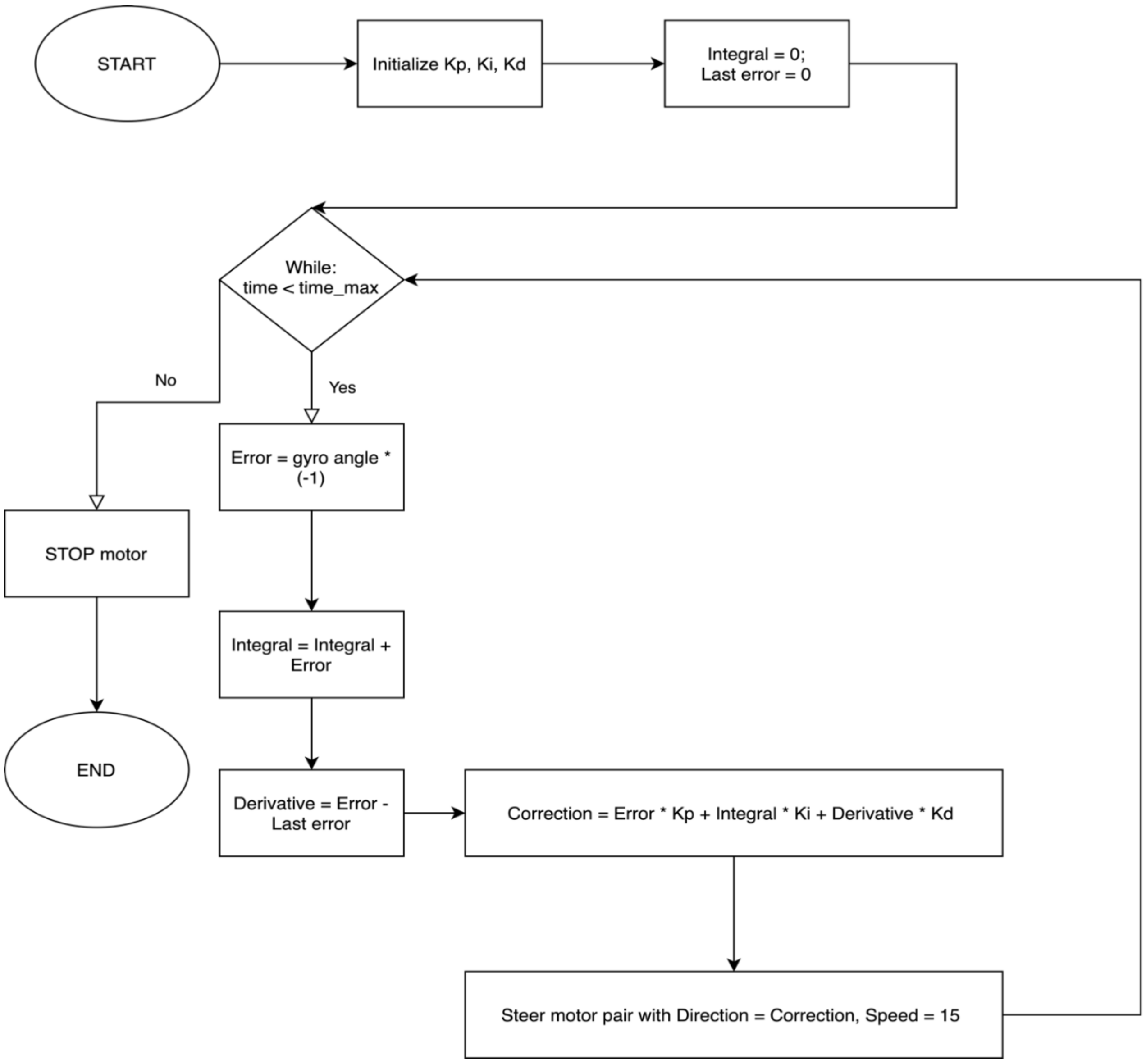

START
Initialize Kp, Ki, Kd
Integral = 0;
Last error = 0
While:
time < time_max
No
Yes
Error = gyro angle * (-1)
STOP motor
Integral = Integral + Error
END
Derivative = Error - Last error
Correction = Error * Kp + Integral * Ki + Derivative * Kd
Steer motor pair with Direction = Correction, Speed = 15

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 flowchart:

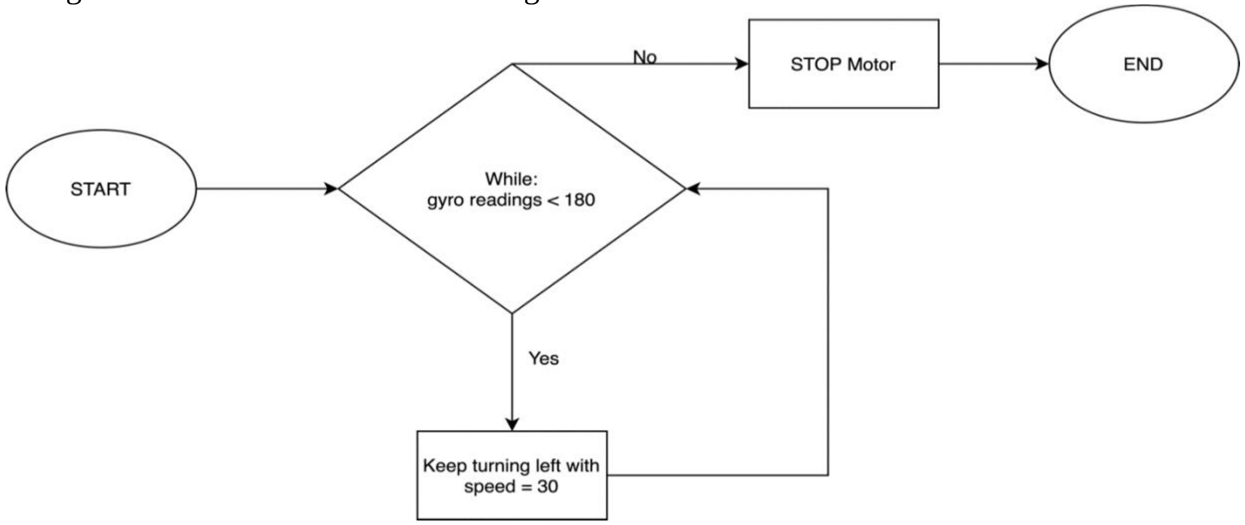
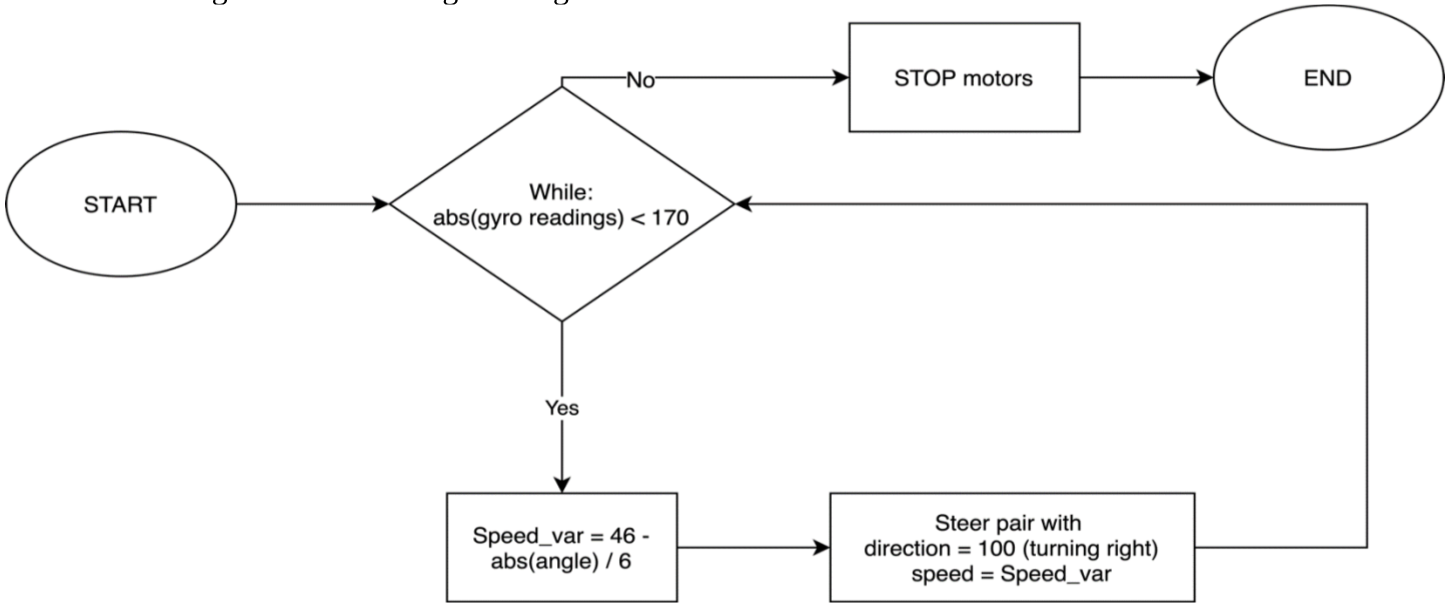


Figure 10: Flowchart for turning 180 algorithm v.1

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:



START
While: abs(gyro readings) < 170
No
STOP motors
END
Yes
Speed_var = 46 - abs(angle) / 6
Steer pair with direction = 100 (turning right) speed = Speed_var

Figure 11: Flowchart for turning 180 algorithm v.2

Our Python code is written as followed:

```python
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)
```

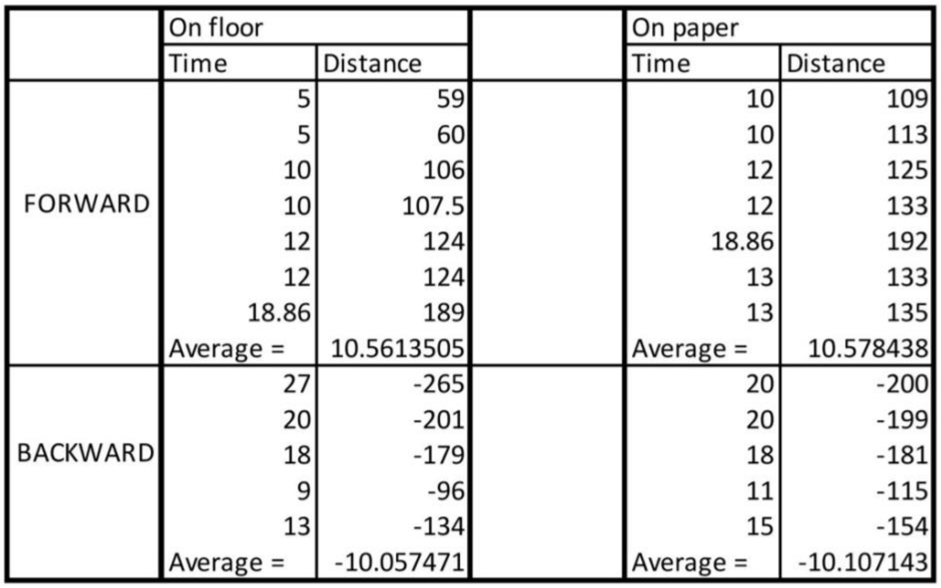
4. Determine the robot's speed:

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:

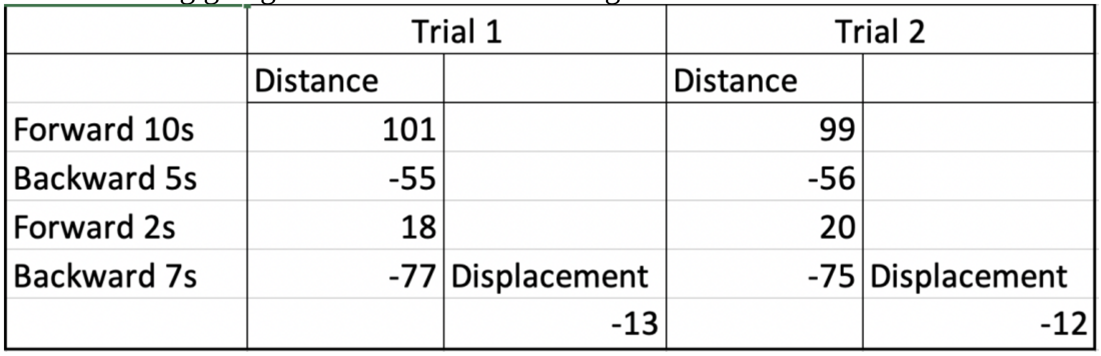


	On floor			On paper	
	Time	Distance		Time	Distance
FORWARD	5	59		10	109
	5	60		10	113
	10	106		12	125
	10	107.5		12	133
	12	124		18.86	192
	12	124		13	133
	18.86	189		13	135
	Average =	10.5613505		Average =	10.578438
BACKWARD	27	-265		20	-200
	20	-201		20	-199
	18	-179		18	-181
	9	-96		11	-115
	13	-134		15	-154
	Average =	-10.057471		Average =	-10.107143

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.

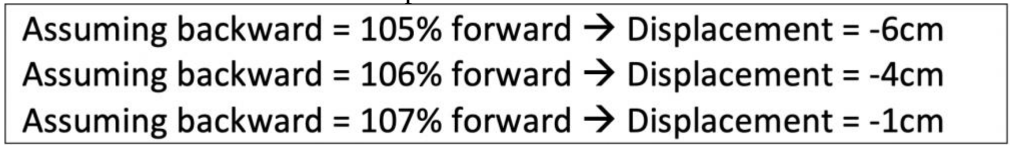
Testing going forward and backward together:



	Trial 1		Trial 2	
	Distance		Distance	
Forward 10s	101		99	
Backward 5s	-55		-56	
Forward 2s	18		20	
Backward 7s	-77	Displacement	-75	Displacement
		-13		-12

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:



Assuming backward = 105% forward → Displacement = -6cm
Assuming backward = 106% forward → Displacement = -4cm
Assuming backward = 107% forward → Displacement = -1cm

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

Subtask 1 demonstration day – 02/18/2020

Feb 18, 2020
Rhodes Hall, 7:30pm – 8pm
Attendance: *Everyone*

Task	Reflection
1. Take robot for demonstration	1. Upon receiving the results of our demonstration, we reflected on our strong and weak points (details below)

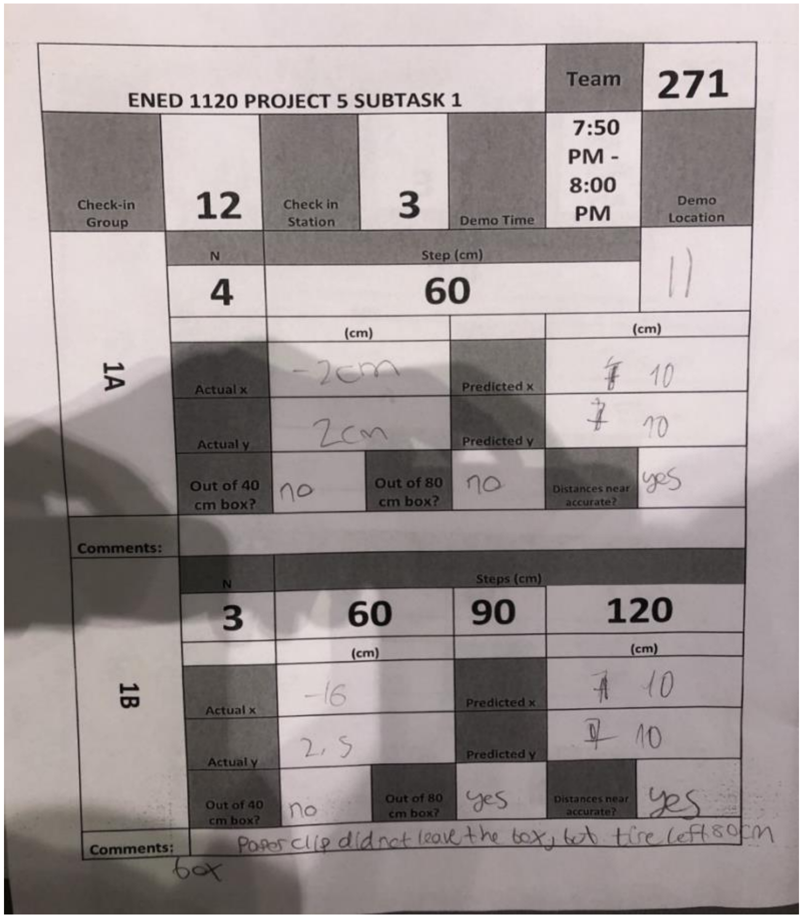
Details:

1. Robot demonstration:

Upon observing the performance of our robot, we deemed our design as 90% successful, as the robot was able to go back and forth (mostly) straight, and the final displacement was much smaller than what we expected.

However, in the turning test of the demonstration, our robot showed much variability and did not turn as well as we tested in on the floor. We figured the reasons could be attributed a new type paper with a new coefficient of friction that we never knew of.

Below is the result of our demonstration



ENED 1120 PROJECT 5 SUBTASK 1						Team	271
Check-in Group	12	Check in Station	3	Demo Time	7:50 PM - 8:00 PM	Demo Location	11

1A	N	Step (cm)				
	4	60				
		(cm)			(cm)	
	Actual x	-2cm		Predicted x	~~7~~ 10	
	Actual y	2cm		Predicted y	~~7~~ 10	
	Out of 40 cm box?	no	Out of 80 cm box?	no	Distances near accurate?	yes
Comments:						

1B	N	Steps (cm)				
	3	60		90	120	
		(cm)			(cm)	
	Actual x	-16		Predicted x	~~7~~ 10	
	Actual y	2,5		Predicted y	~~7~~ 10	
	Out of 40 cm box?	no	Out of 80 cm box?	yes	Distances near accurate?	yes
Comments:	Paper clip did not leave the box, but tire left 80cm box					

Figure 15: Results of team 271's subtask 1 demo

Meeting 3: Pickup system – 02/19/2020

Feb 19, 2020
Langsam Library, 4pm – 6pm
***Attendance:** Everyone*

Task	Reflection
2. Critique and improve robot design from the performance of the demo 3. Deciding on a lifting mechanism for the robot 4. Building the prototype lifting system 5. Test the functionality	2. We added support for the back, reinforced the robot's frame on the bottom layer, and refined the turning algorithm (details below) 3. With a decision matrix' help, we agreed on a hook and lift system to pick the box (details below) 4. We built the hook system with conveyor belt and a protruding lifting arm (details below) 5. 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:

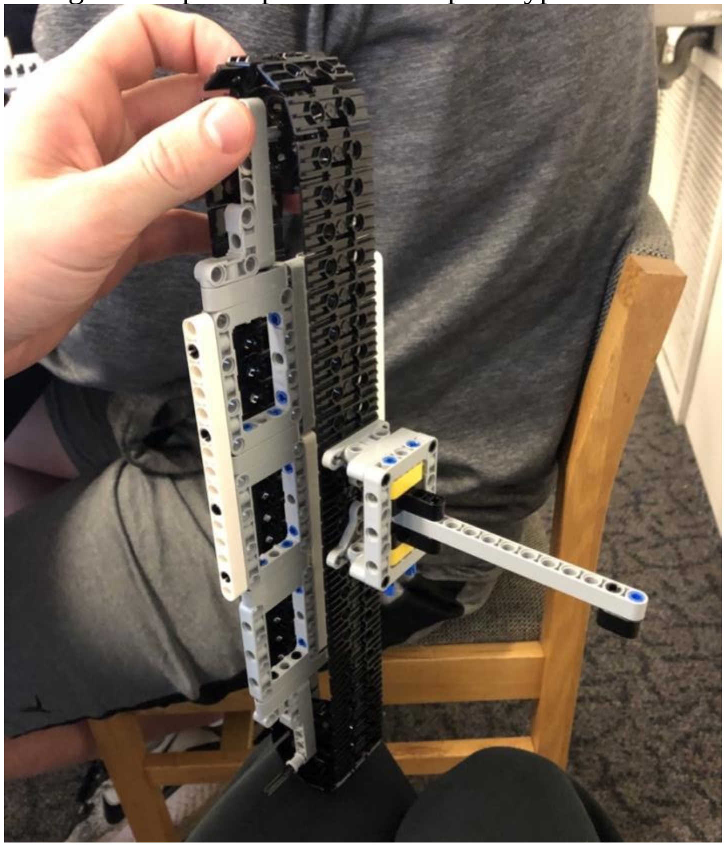


Figure 16: Conveyor belt frame + pickup arm for the lift system

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:

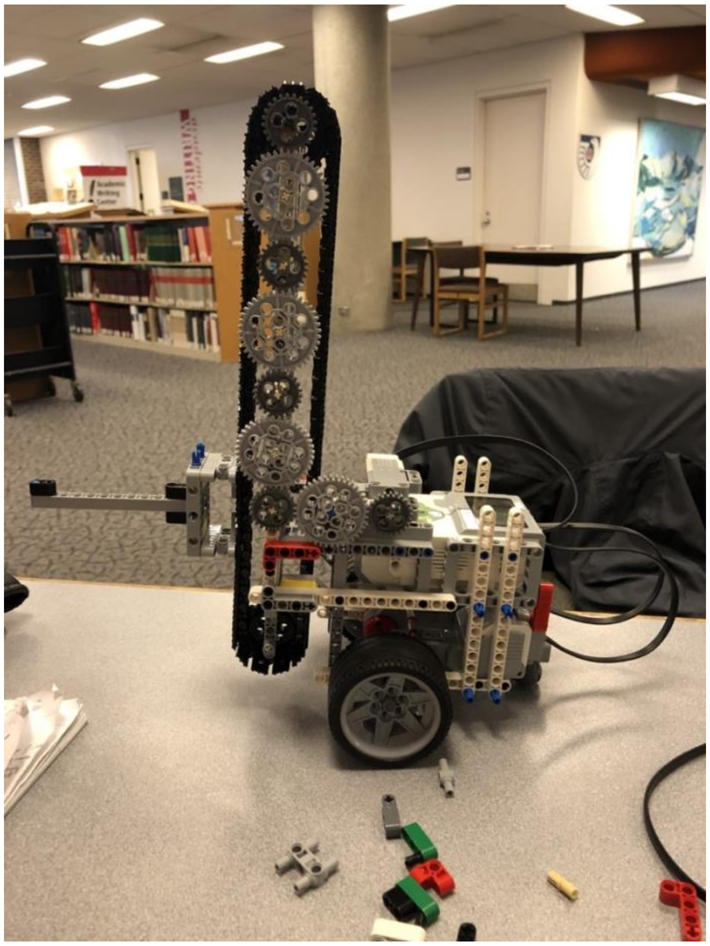


Figure 17: Final design for robot's lifting mechanism

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) 2. We installed a color sensor to allow the robot to read the barcode on the box (details below) 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)

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.

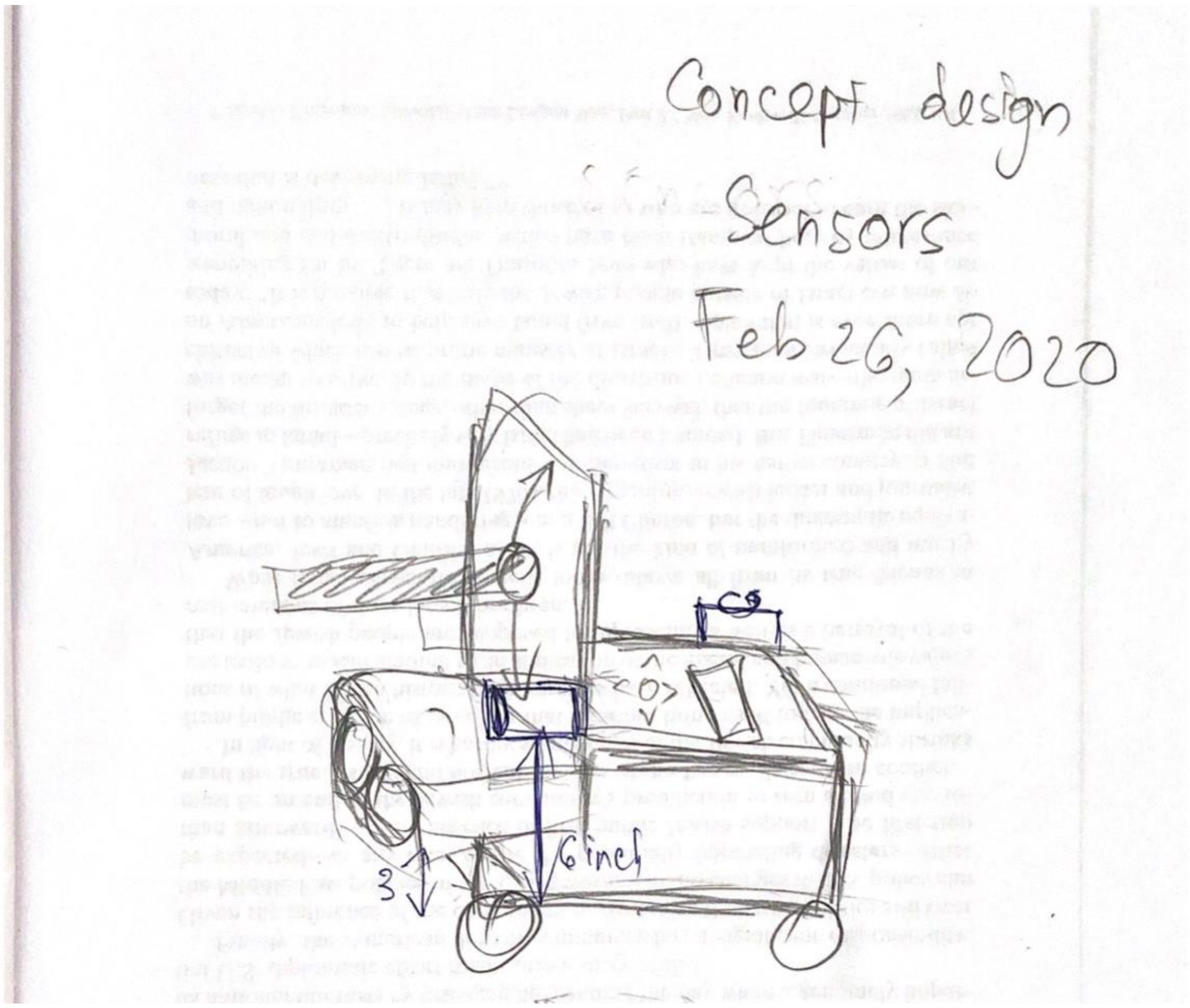


Figure 18: Concept design for sensors layout

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) 2. We constructed an algorithm allowing the robot to navigate through shelves and across quads in the arena to check boxes (details below) 3. We constructed a back-track scanning algorithm to allow the robot to stop and scan barcode when robot encounters a box (details below)

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.

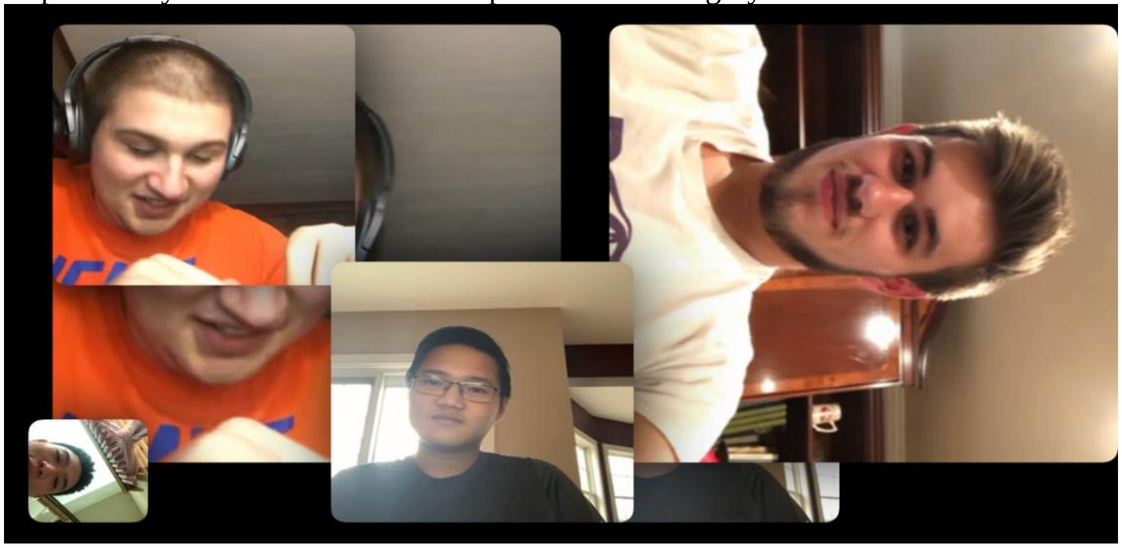


Figure 19: Image of team 271 videoconferencing

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:

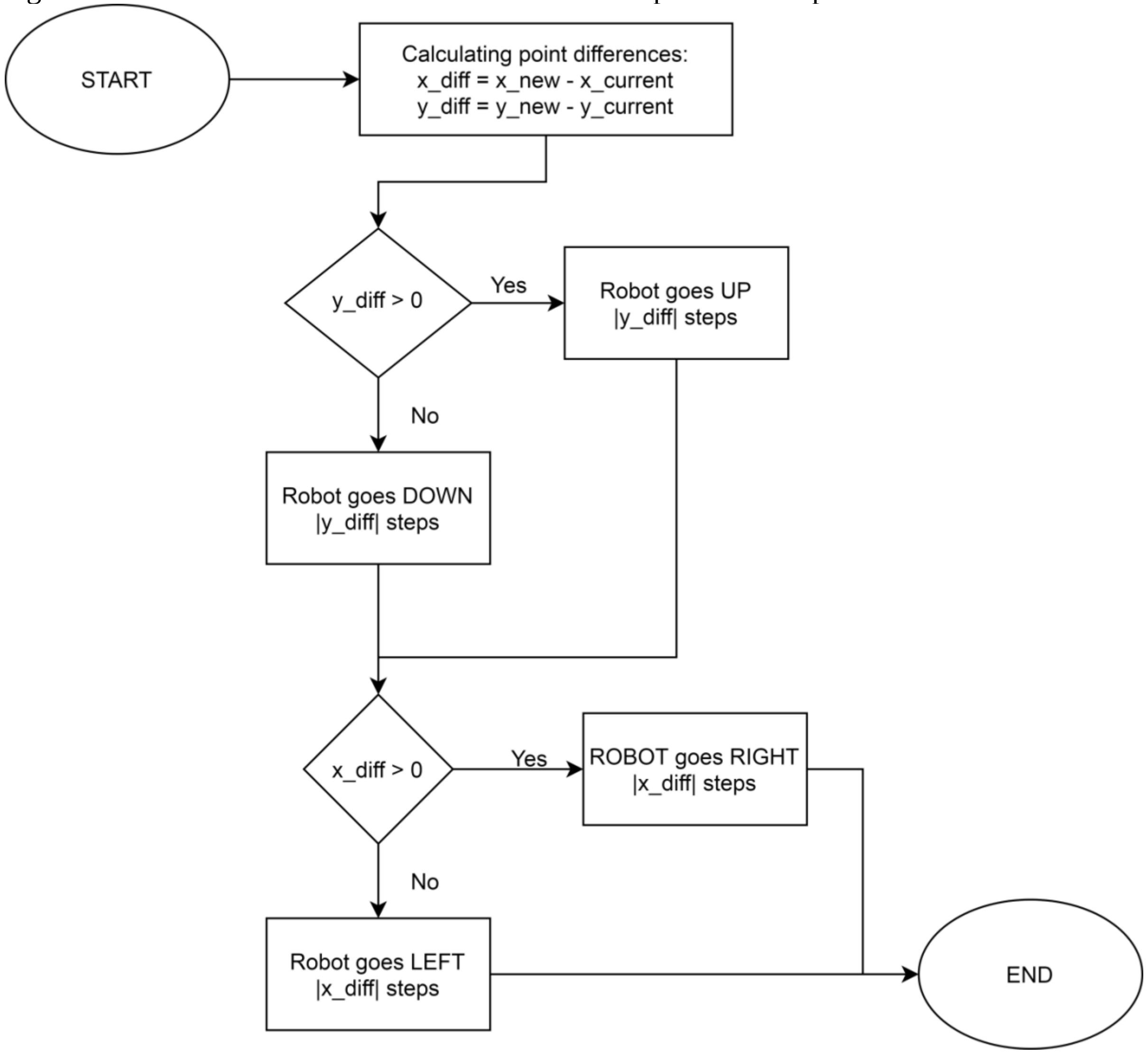


Figure 20: Flowchart for algorithm to navigate between 2 points

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:

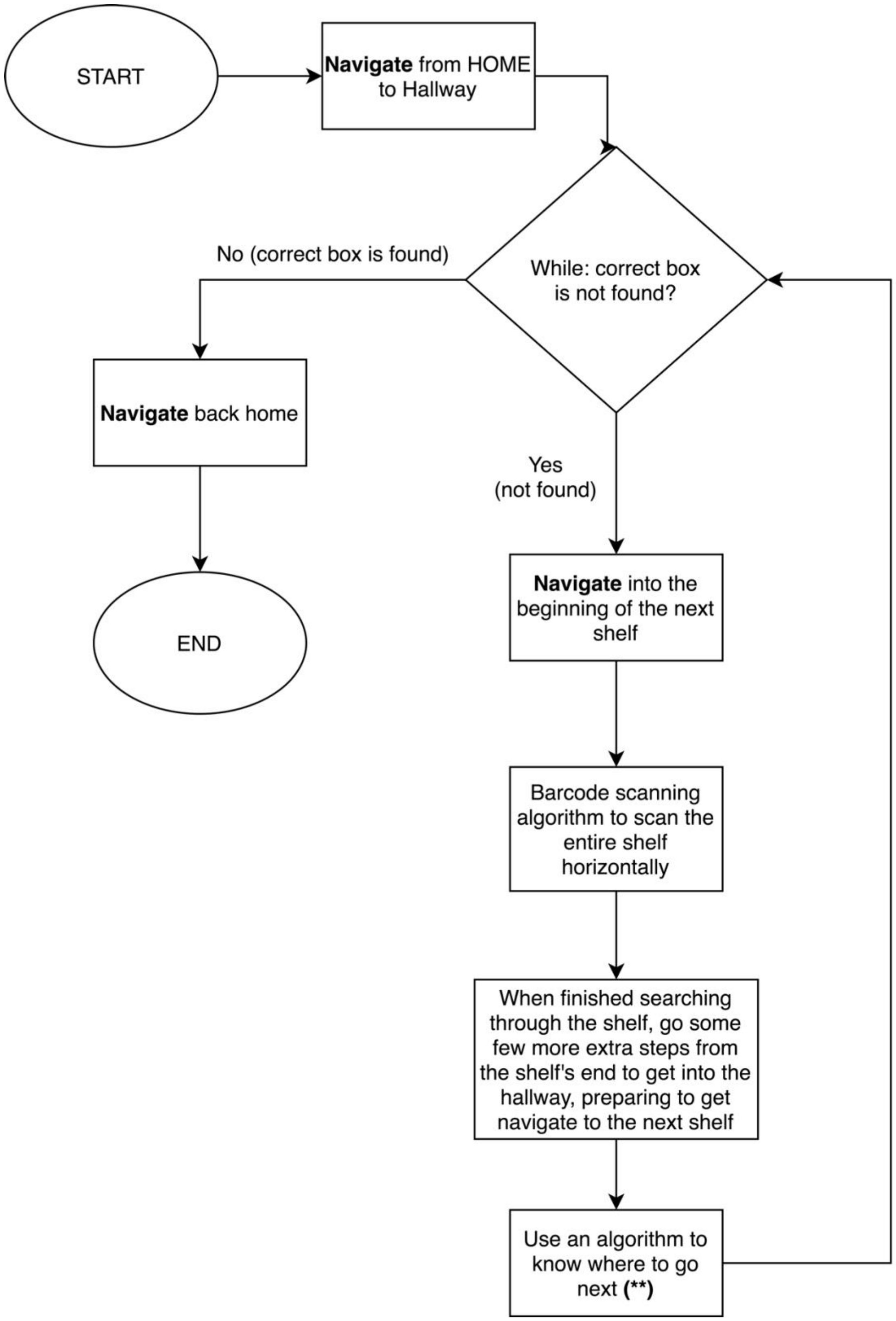

START
Navigate from HOME to Hallway
While: correct box is not found?
No (correct box is found)
Navigate back home
END
Yes (not found)
Navigate into the beginning of the next shelf
Barcode scanning algorithm to scan the entire shelf horizontally
When finished searching through the shelf, go some few more extra steps from the shelf's end to get into the hallway, preparing to get navigate to the next shelf
Use an algorithm to know where to go next **(**)**

Figure 21: 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:

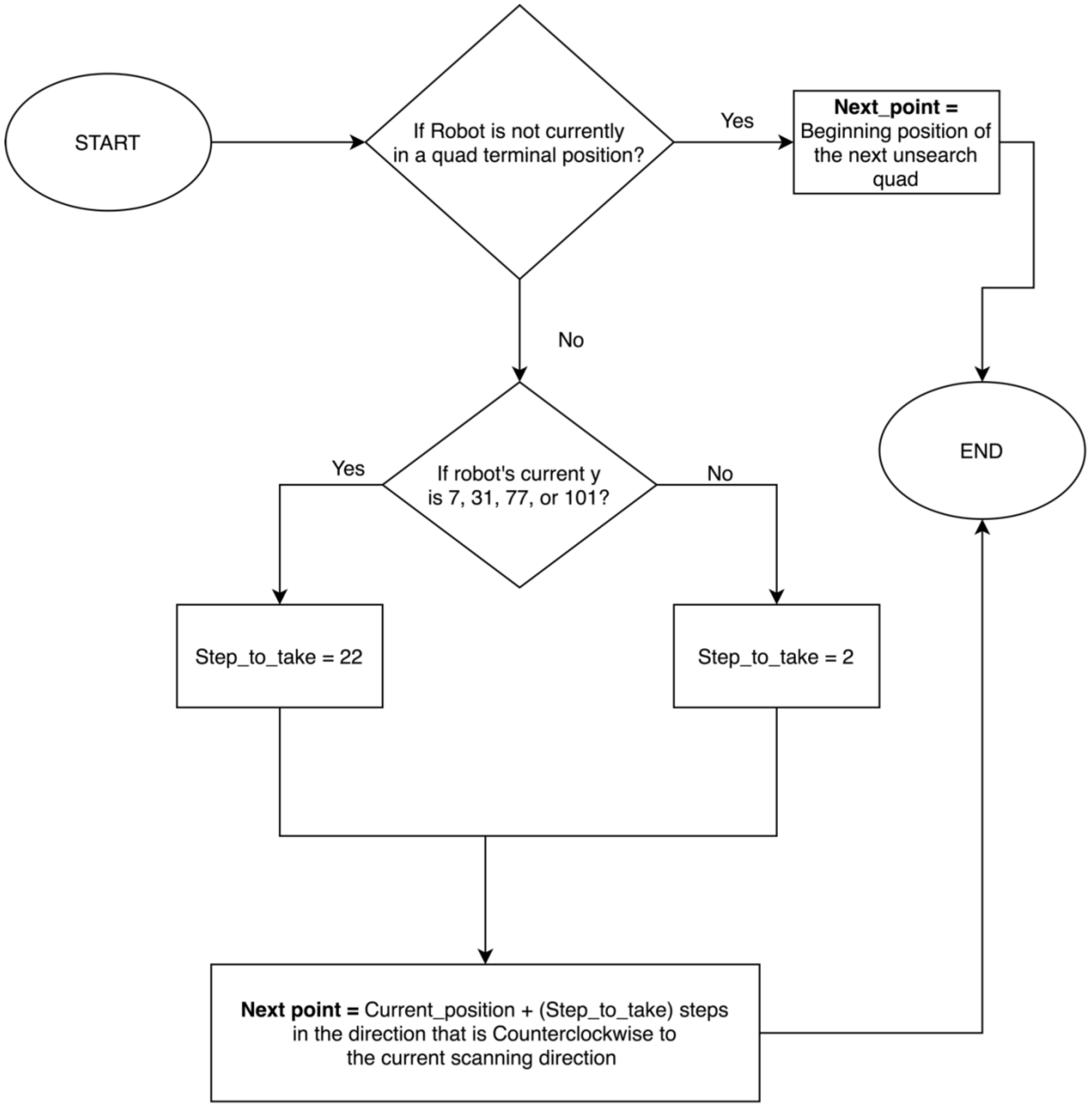

START
If Robot is not currently in a quad terminal position?
Yes
Next_point = Beginning position of the next unsearch quad
No
END
If robot's current y is 7, 31, 77, or 101?
Yes
No
Step_to_take = 22
Step_to_take = 2
Next point = Current_position + (Step_to_take) steps in the direction that is Counterclockwise to the current scanning direction

Figure 22: 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:

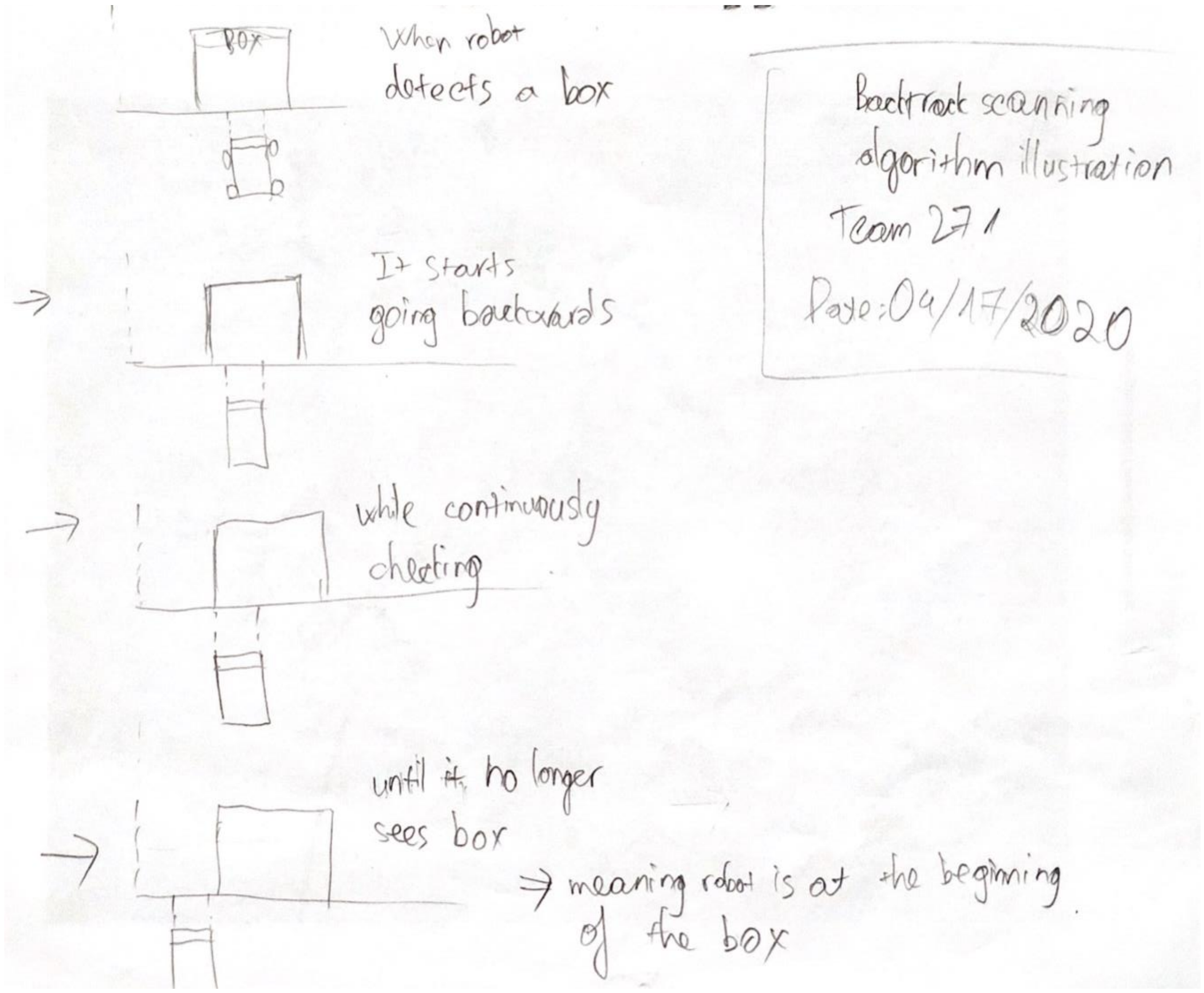

BOX
When robot detects a box
Backtrack scanning algorithm illustration
Team 271
Date: 04/17/2020
It starts going backwards
while continuously checking
until it no longer sees box
→ meaning robot is at the beginning of the box

Figure 23: Illustration for backtrack-scanning algorithm

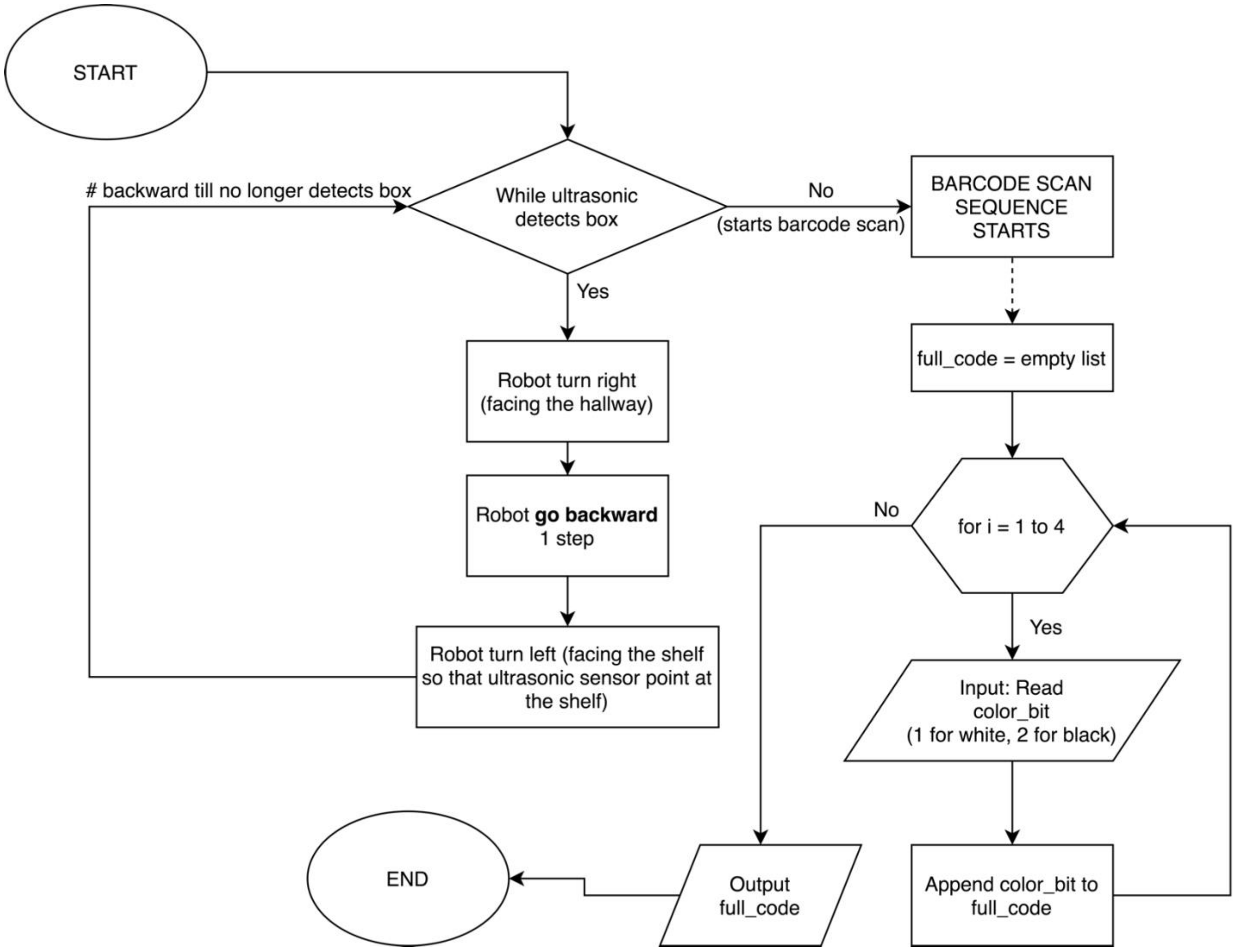

START
While ultrasonic detects box
# backward till no longer detects box
No
(starts barcode scan)
BARCODE SCAN SEQUENCE STARTS
Yes
Robot turn right (facing the hallway)
Robot go backward 1 step
Robot turn left (facing the shelf so that ultrasonic sensor point at the shelf)
full_code = empty list
for i = 1 to 4
No
Yes
Input: Read color_bit (1 for white, 2 for black)
Append color_bit to full_code
Output full_code
END

Figure 24: 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. Develop our code for the simulation 2. Film videos for subtask 2	1. We developed our simulation code in Python, which includes visual representation of the robot and boxes, and navigation + barcode scan algorithm. (details below) 2. We filmed 2 videos on our physical and logical progress and submitted them on BlackBoard (details below)

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.

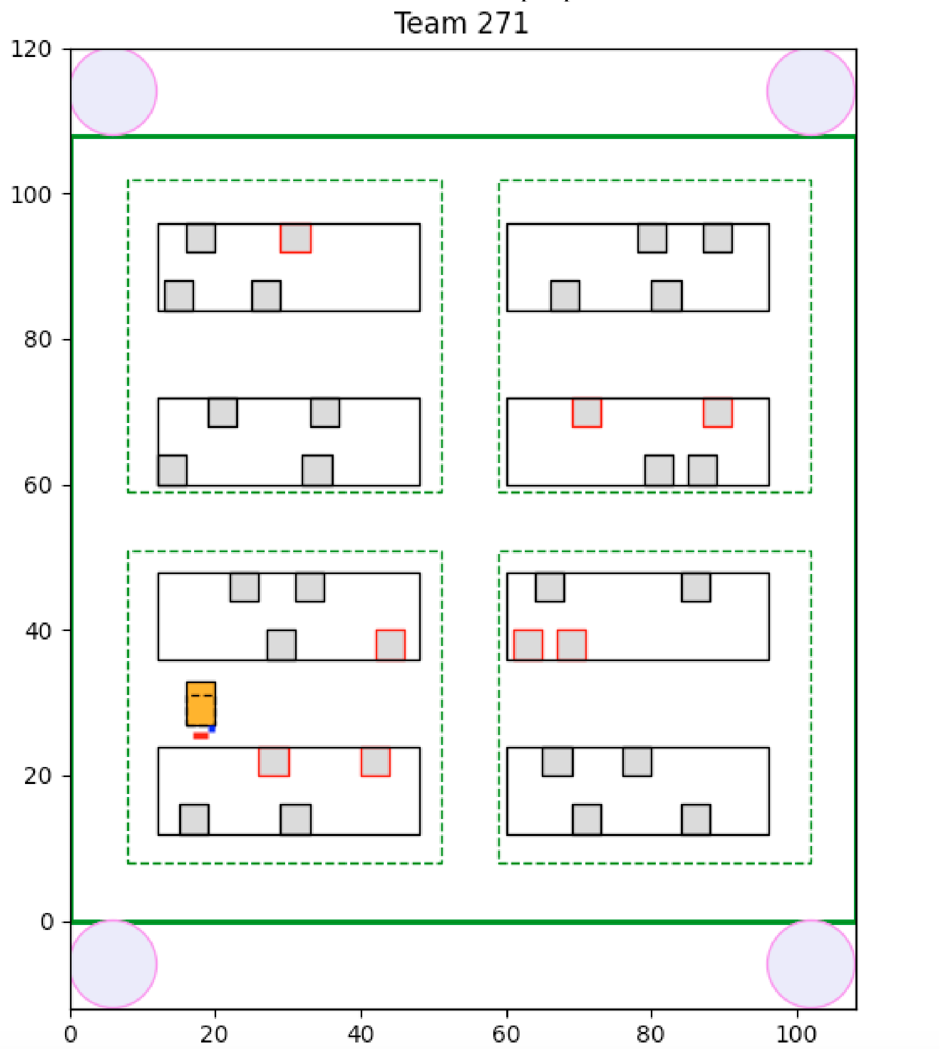


Figure 25: Visual representation of team 271's simulation

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

```python
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:

```python
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) 2. We brainstormed a collision avoidance algorithm to circumvent stationary obstacles (details below) 3. We added the collision avoidance code into our simulation (details below)

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:

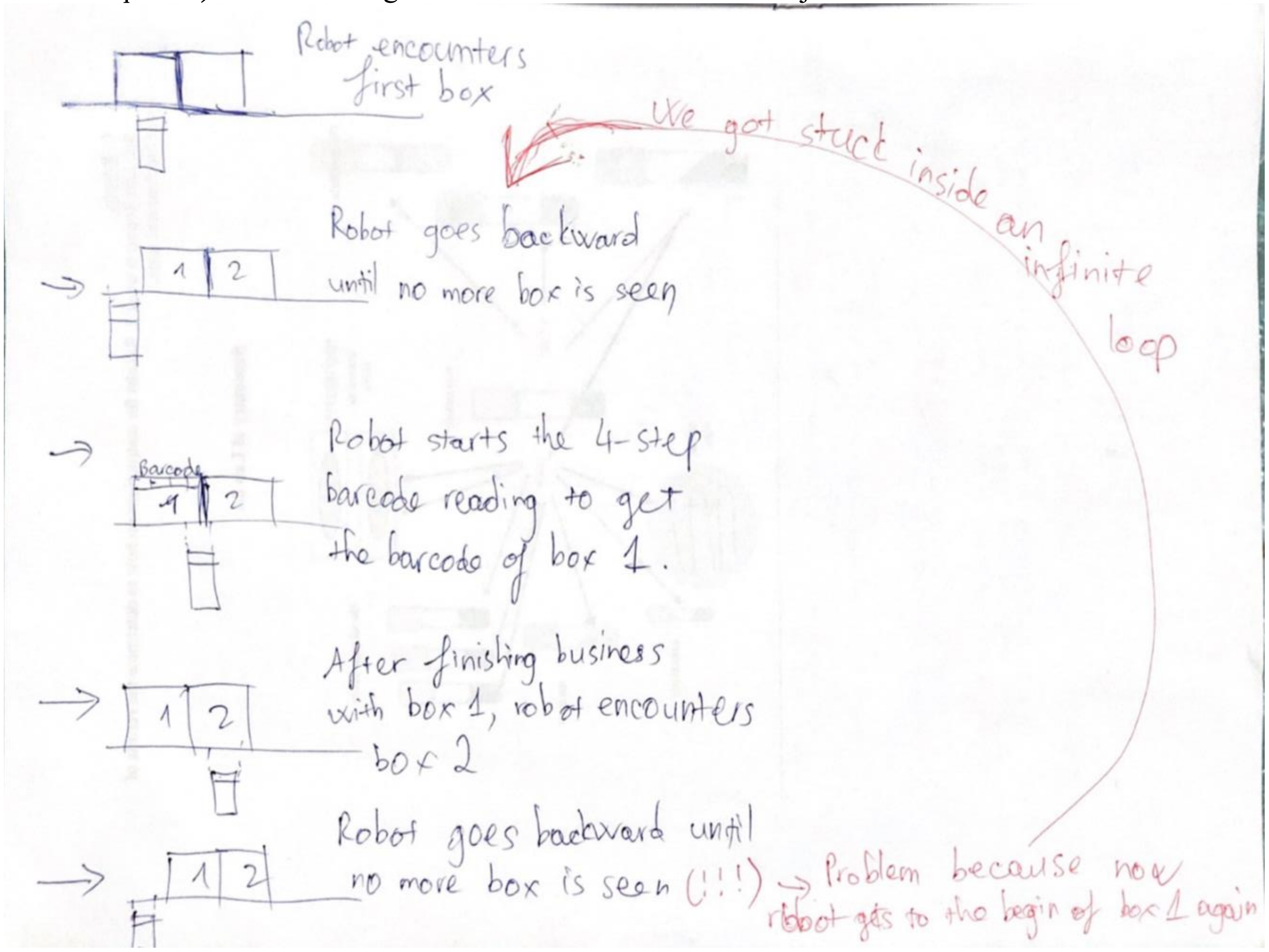
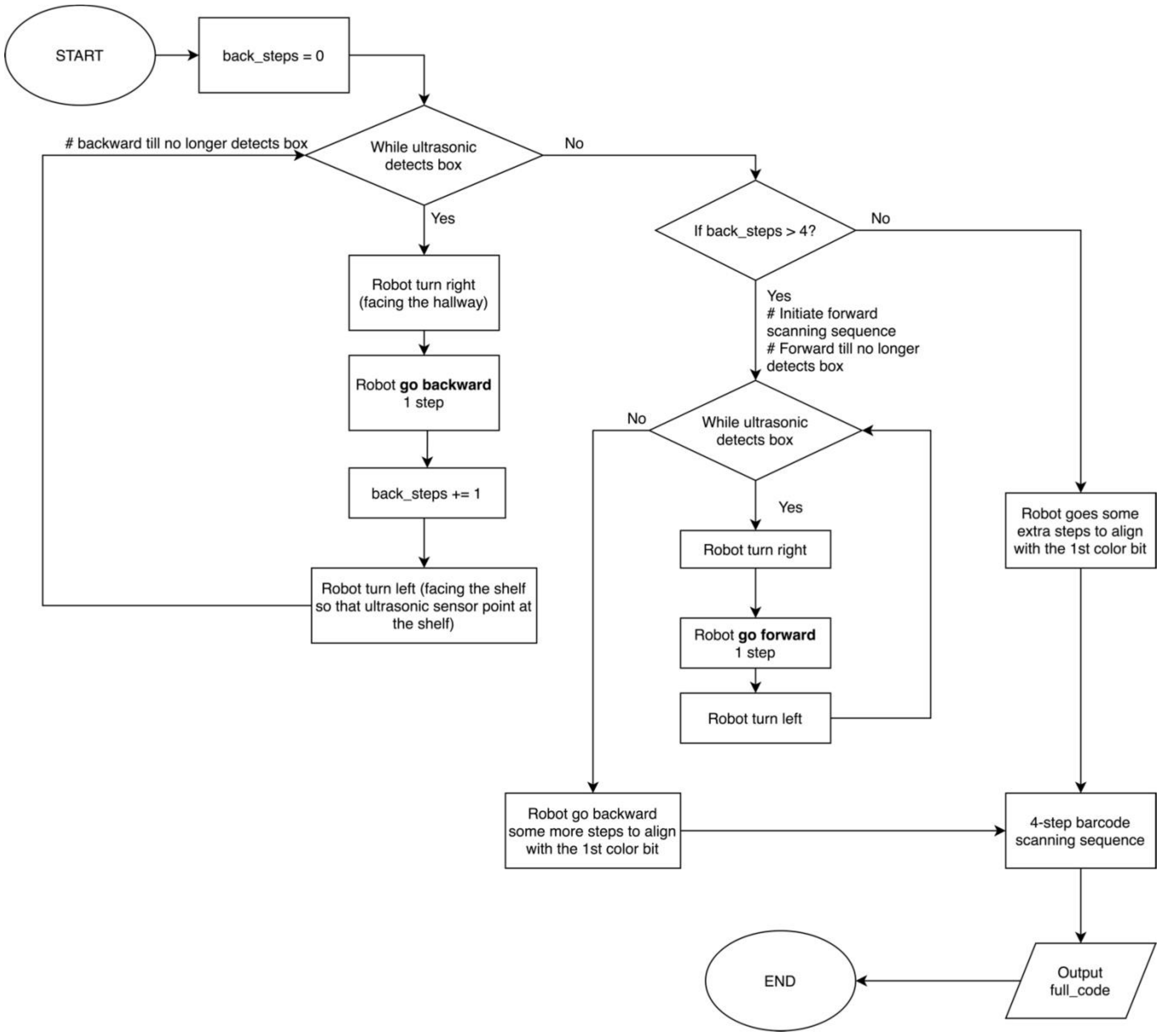

Robot encounters first box
We got stuck inside an infinite loop
Robot goes backward until no more box is seen
1
2
Robot starts the 4-step barcode reading to get the barcode of box 1.
Barcode
After finishing business with box 1, robot encounters box 2
Robot goes backward until no more box is seen (!!!) → Problem because now robot gets to the begin of box 1 again

Figure 26: 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:



START
back_steps = 0
# backward till no longer detects box
While ultrasonic detects box
No
Yes
Robot turn right (facing the hallway)
Robot **go backward** 1 step
back_steps += 1
Robot turn left (facing the shelf so that ultrasonic sensor point at the shelf)
If back_steps > 4?
No
Yes
# Initiate forward scanning sequence
# Forward till no longer detects box
While ultrasonic detects box
No
Yes
Robot turn right
Robot **go forward** 1 step
Robot turn left
Robot goes some extra steps to align with the 1st color bit
Robot go backward some more steps to align with the 1st color bit
4-step barcode scanning sequence
Output full_code
END

Figure 27: Flowchart for backtrack scanning algorithm v.2

The following is the Python code for our refined backtrack scanning algorithm:

```
def backtrack_scanning(self):
        """
        Backtrack scanning algorithm when robot has met a box
        """
        scanning_direction = self.get_scanning_direction()
```

```
    # 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
        backward_steps = 0 # steps count to control how long robot has backwarded
        while self.robot.ultrasonic_detection(self.backend.board):
            backward_steps += 1
            self.robot_turn_right()
            self.robot_backward(1)
            self.robot_turn_left()
            plt.pause(0.02)
        print(f"Back steps: {backward_steps}")
        # If it needs to go backward more than 4 steps, there are 2 boxes
        # next to each other: The "adjacent boxes" problem
        # Therefore, activate "forward-tracking scan" sequence
        if backward_steps > 4:
            print("Adjacent boxes found")
            # Forward 10 steps to get into a position that aligns with box II
            self.robot_turn_right()
            self.robot_forward(10)
            self.robot_turn_left()
            # Forward till no longer see box
            while self.robot.ultrasonic_detection(self.backend.board):
                self.robot_turn_right()
                self.robot_forward(1)
                self.robot_turn_left()
                plt.pause(0.02)
            # When the box is no longer seen, the ultrasonic field is at the box's
            # rightmost edge. Now turn right, preparing to backward to scanning position
            # Visual Bug: If robot is going RIGHT, needs to backward 3 steps for
            # the colorsensor to align with the first barcode bit of box II
            # If robot is going LEFT, needs 4 steps instead
            self.robot_turn_right()
            self.robot_backward(3 if scanning_direction == RIGHT else 4)
        else:
            # If there is one box isolated, there are no concerns
            # 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 is going RIGHT, needs to forward 4 steps for
            # the colorsensor to align with the first barcode bit
            # If robot is going LEFT, only needs 3 steps
            self.robot_forward(4 if scanning_direction == RIGHT else 3)
        self.robot_turn_left()
        plt.pause(0.05)
        full_code = self.scan_full_barcode()
        print(f"Barcode result: {full_code}")
        plt.pause(0.5)
        return full_code
```

2. Collision Avoidance system:

Our Collision Avoidance system (abbreviated as CAS from now on) utilizes an ultrasonic sensor to detect if there are obstacles nearby. Due to limited instructions regarding how the obstacles will be set up, for our CAS to be effective, our team scoped the difficulty of the project and decided to make some reasonable assumptions about the obstacles (will be called “rocks” from now on) that the robot will encounter:

- The rocks will not block the entrance into any shelves. If any rock blocks the entrance into a shelf, it will severely affect the robot’s ability to retrieve the barcode, which our team determined to be against the goal of this project: to retrieve a box. Thus, no rocks should appear in the inter-shelf area, or it may the path to scan and collect boxes.
- The obstacles (will be called “rocks” from now on) is at most size 4. If it is too large, it has very high chance at blocking many paths entirely.
- The rocks will be stationary. Due to time constraints and the limit of our team’s capability, we can only circumvent stationary rocks. Therefore, we will not simulate moving obstacles
- The rocks’ frequency shall not be high, that is, not too many rocks are in one place, or they will inevitably clog the paths.

With the assumptions above. We implemented two supplementing strategies to avoid as much collision as possible.

First, we will reduce the frequency of robot colliding with the box by programming the robot to go very close the shelf borders. Because rocks have higher chance to appear on the hallway, we think this is a valid strategy for collision avoidance.

Secondly, when the first strategy fails to eliminate all the rocks the robot has to meet, we implement a circumvention algorithm to go around the rocks while keeping the original path. When the robot meets a rock, it will start sidestepping one step at a time, then turn back and use the ultrasonic sensor to see if the rock is still there; if yes, it will continue side stepping, and ultrasonic-check again, and loop that until the rock no longer block its path. Then it will go forward in order to “overtake” the rock, finally sidestepping back in to get back to the original path.

The direction that the robot will sidestep is determined by its coordinate. For example, if the robot is on column 7 or 58, it should sidestep to the LEFT.

3. Collision Avoidance code:

The Python code for our rock circumvention is as followed:

```python
def get_dodging_direction(self):
        """
        Return the dodging direction when meeting a rock
        """
        current_col = self.center[0]
        if (6 <= current_col <= 15 or 54 <= current_col <= 65):
            return LEFT
        else:
            return RIGHT
```

```python
def circumvent_rock(self):
        """
        Go around an obstacle
        """
        dodge_direction = self.robot.get_dodging_direction()
        print(f"Doding direction is: {dodge_direction}")
        original_direction = self.robot.direction
        rock_size = 0
        # Robot dodges to the side until there are no more rock blocking way
        while self.robot.ultrasonic_detection(self.backend.board):
            self.robot_become_direction(dodge_direction)
            self.robot_forward(1)
            rock_size += 1
            self.robot_become_direction(original_direction)
            plt.pause(1)

        # When there are no more rocks blocking way, robot does forward
        # How many steps dodged, that many steps forward, because rock is square shape
        self.robot_forward(rock_size)
        # Robot turn back into original path
        self.robot_become_direction(rev(dodge_direction))
        self.robot_become_direction(original_direction)
```

Meeting 8: The IPS system and Final submissions

April 24, 2020
Virtual texting, 10am – 10:30am
Attendance: *Everyone*

Task	Reflection
1. Develop the IPS system 2. Write report 3. Film demonstration video	1. For the IPS, we inputted coordinates and distances information onto the screen the demonstrate the IPS (details below) 2. We wrote the report summarizing our progress (details below) 3. We filmed a 6-minute video to demonstrate logical components (details below)

Details

1. Indoor Positioning system simulation:

For the simulation of our IPS system, we decided to write text onto screens to illustrate some of the calculations we made for the IPS, which includes:

- Simulation coordinates: This is the coordinate encoded in our simulation code
- Radial distances: Radial distances from HOME A, C, and D. We calculate the distances between 2 points using the formula: $distance = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$
- Calculated coordinates: From the radial distances, we calculate the IPS coordinates using the IPS algorithms given in the RFAI 1:

$x = (-(y_2 - y_3)*((y_2\text{^}2-y_1\text{^}2)+(x_2\text{^}2-x_1\text{^}2)+(r_1\text{^}2-r_2\text{^}2)) + (y_1-y_2)*((y_3\text{^}2-y_2\text{^}2)+(x_3\text{^}2-x_2\text{^}2)+(r_2\text{^}2-r_3\text{^}2)))/(2*((x_1-x_2)*(y_2-y_3) - (x_2-x_3)*(y_1-y_2)))$

$y = (-(x_2 - x_3)*((x_2\text{^}2 - x_1\text{^}2) + (y_2\text{^}2 - y_1\text{^}2) + (r_1\text{^}2 - r_2\text{^}2))+((x_1-x_2)*((x_3\text{^}2-x_2\text{^}2)+(y_3\text{^}2-y_2\text{^}2)+(r_2\text{^}2-r_3\text{^}2))))/(2*((y_1-y_2)*(x_2-x_3) - (y_2-y_3)*(x_1-x_2)))$

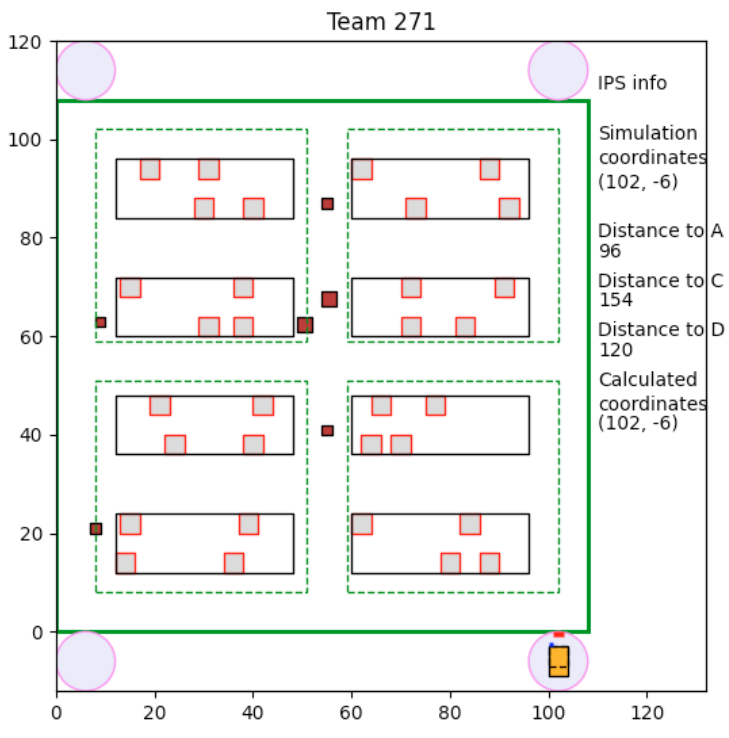


Figure 28: Image of team 271's IPS simulation

With the IPS system successfully constructed and tested, it is time we finalized our project 5 and submit our work.

2. Design process report:

We wrote our design process report to summarize and reflect on how we have done during project 5. One main points we all agreed on was how strong commitment to make consistent weekly progress, which helped us boost the completion of the physical component significantly.

However, one of the weak points we found was our lack of proper testing for our prototypes. We were quick to build, but often neglected the testing phase. Therefore, some of our prototypes needed some rebuild or extra reinforcements. We believed that was because we did not test the robot under a diverse range of situation and therefore, was not able to control the variability.

Another thing we want to improve in the future is our far-vision which will help us develop prototypes that not only works well for its own functionality but also integrates with future systems yet to be built. Often, we found ourselves extending all the spaces available to build one system, and then when the next system was needed to be built, we had to revamp the robot in order the create more spaces or wires or sensors to fit in. Therefore, from the experiences of project 5, we think that in the future, when we ideate one system, we should brainstorm about how a particular design will integrate with other components better as well.

We also calculated the personnel cost and project cost for our project 5:

1. Bao Huynh

- Roles: Team leader, Timekeeper, Quality control, Notebook editor, Main programmer
- Hours spent: 50 hours
- Personnel cost: $40 / hour * 50 hours = $2000

2. Triet Pham

- Roles: Assistant builder, Assistant programmer, Notebook editor
- Hours spent: 40 hours
- Personnel cost: $40 / hour * 40 hours = $1600

3. Nathan Damas:

- Roles: Main builder, Notebook editor
- Hours spent: 40 hours
- Personnel cost: $40 / hour * 40 hours = $1600

4. John Cummings:

- Roles: Assistant builder, Main engineering notebook recorder
- Hours spent: 40 hours
- Personnel cost: $40 / hour * 40 hours = $1600

➔ **Total project cost: $2000 + $1600 + $1600 + $1600 = $6800**

3. Demonstration video:

We filmed a 6-minute video to demonstrate to the technical team at Harris Corporation. In our video, we showed that our simulation has sufficient IPS information, and that the robot can navigate through the entire field to search for the box while also avoiding obstacles. We also explained our backtrack-scanning box search algorithm along with visual representation. Our video has 2 main parts – The first part showed our robot navigate through the entire field by inputting an invalid barcode (the robot will scan the whole area and return “No box found” for the search result) and the second part showed the robot is capable of picking up the right box with the correct barcode. In both the trials, the IPS and collision avoidance systems were always active.

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