MIE444 Rover Project Final Report

Chen Hsin Chiang 1003888484
Jeongwoong Choi 1004942743
Lijin Yu 1004720901
Yunxiang Zhang 1005085765

December 15, 2022

Department of Mechanical and Industrial Engineering
Faculty of Applied Science and Engineering
University of Toronto

1. Executive Summary

In the first Milestone, the team implemented a simple obstacle avoidance technique to turn the rover towards the side with the greatest distance. However, the team discovered that localization required the rover to travel straight. Therefore, obstacle avoidance logic was improved.

By Milestone 3, the rover was able to avoid obstacles, travel a straight path, correct heading and localize in a single trial. It would update its heading correctly after reaching the 4-way intersection. Next, it was able to recognize its location based on the localization logic. Using the localization algorithm, the rover followed pre-written paths to the loading/unloading zone. On one hand, the rover was able to avoid obstacles with perfection, on the other hand, the rover was unable to complete all the tasks within the required time.

In the second trial, the team only demonstrated the load-finding and pick-up functions. With the initial placement of the rover at the exterior of the loading zone, the rover was able to traverse into the loading zone with minor help and detect the load. Following the detection of the load, the rover adjusts its position to pick up the load. However, due to the poor design of the gripper mechanism, it was unable to lift the load. Based on the observed movements of the servo motors, the gripper was outputting the right commands. With the appropriate servo motor selection and enough power supply, the gripper would work as intended.

For the final reflection, the most significant area of improvement was time management. Because of poor time allocation, many deliverables were rushed at the last minute. For instance, rover & gripper design, sensor adjustment and tuning, and integration of the rover's code.

Video footage of Milestone 3 can be viewed from the link below:
https://drive.google.com/file/d/18nUjuD-q-UtoxJ-DPNPjeCtA-5GGfJnk/view?usp=share_link

2. Detailed Rover Control Strategy

2.1. Obstacle Avoidance Strategy

Since only four heading directions (W, A, S, D) were defined in the 2D IR localization algorithm, it was crucial that the robot travelled in a relatively straight path while avoiding obstacles. If the rover regularly travelled along the diagonal direction, the probability matrix did not update accurately, and the localization did not work well. Therefore, one of the major objectives for the final obstacle avoidance strategy was to ensure travelling straight.

For obstacle avoidance, SimMer was first utilized for proof of concept: The team first conducted simulations with lower sensor error settings to ensure the validity of the logic. Once all obstacle avoidance objectives were achieved, the team then converted codes to be compatible with Arduino and Bluetooth and tested them in the real maze. To achieve travelling straight, the team first attempted the below sensor setup:

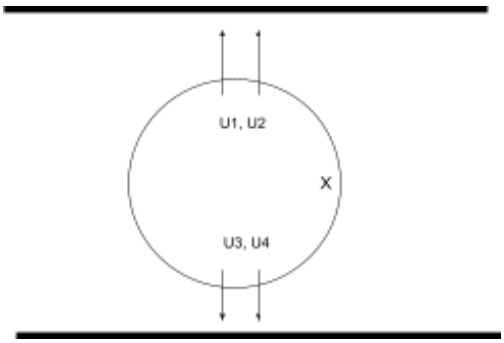

Figure 1: Sensor Configuration for Obsolete Approach

It was conjectured that when the rover travelled in a straight path, the difference in reading between U1&U2 and U3&U4 would be small. Thus, if the difference in reading on either side exceeded a threshold, we could adjust the rover by rotating itself until the difference values are within the threshold. However, this method did not work well, even in the simulation, due to inaccuracies in sensor measurements. When the sensors were at an angle from the wall, they often returned inaccurate distances, (could be higher or lower than expected), which made the difference calculations unreliable. The team could not find a single threshold value that accounted for the inaccuracies while keeping the robot travelling straight. Thus, this approach was abandoned.

The other approach, which was included in the eventual obstacle avoidance strategy, chose the side sensor that returned a smaller reading at the current step (U3) and compared the reading with the one at the previous step (U3’). If U3 < U3’ (as illustrated

in Figure 3), the robot would rotate a small angle counterclockwise to correct the angle and compensate for the deviation. If U3 > U3', the rover would rotate clockwise.

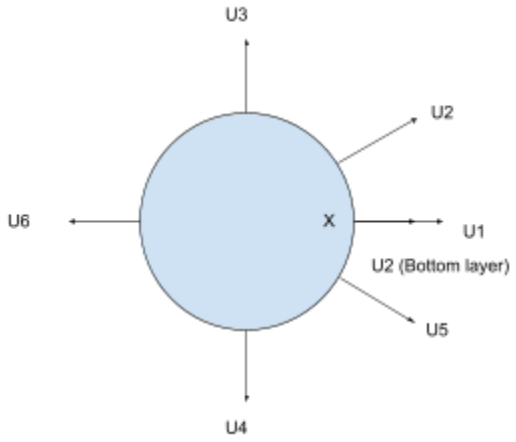

Figure 2. Final Sensor Configuration of the Rover

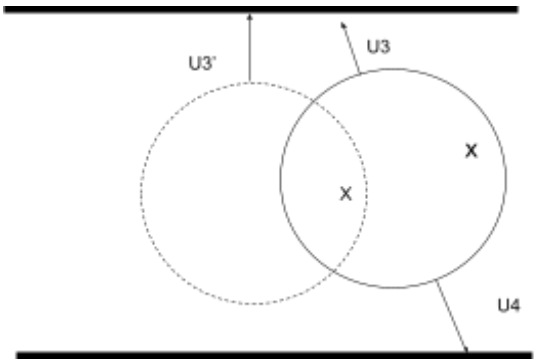

Figure 3. Distance Comparison between the Current and the Previous Step

Once the implementation of the above angle correction logic in SimMer was successful, the team integrated the code with the obstacle avoidance strategy used in Milestone 1. The final strategy can be summarized in the flowcharts below.

As the last step of simulation code testing, the strategy was packaged into MATLAB functions and called during each loop of the main function. At this point, obstacle avoidance was running in an infinite loop. This strategy still performed well after the code was converted from the simulation to the real robot application. The team only needed to modify the sensor threshold values based on the real maze, which determined if the rover was too close to the wall. To establish all sensor threshold values, the team took ultrasonic sensor measurements at various locations in the maze. This obstacle avoidance strategy allowed the rover to run indefinitely without hitting a wall.

Angle Correction Logic:

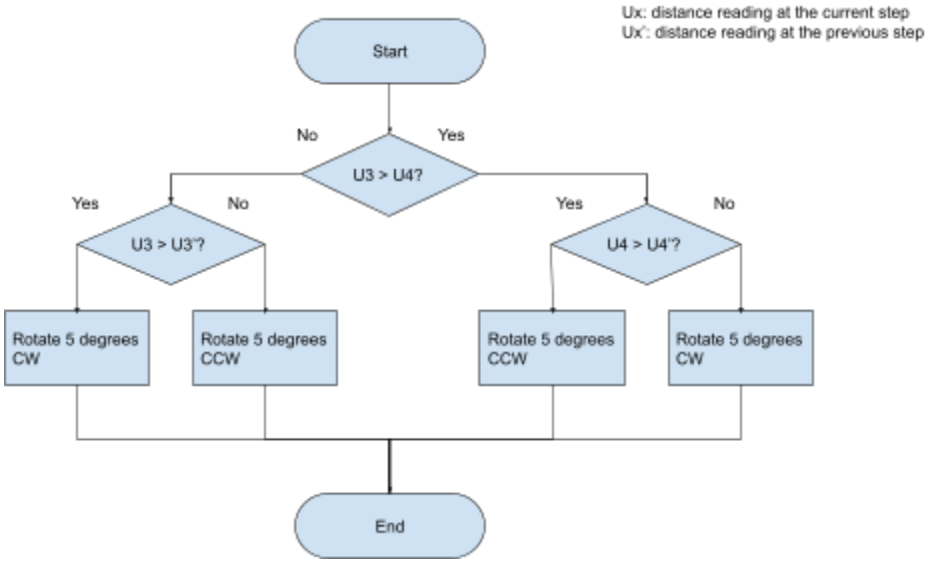

Figure 4. Angle Correction Logic Flowchart

Obstacle Avoidance Logic:

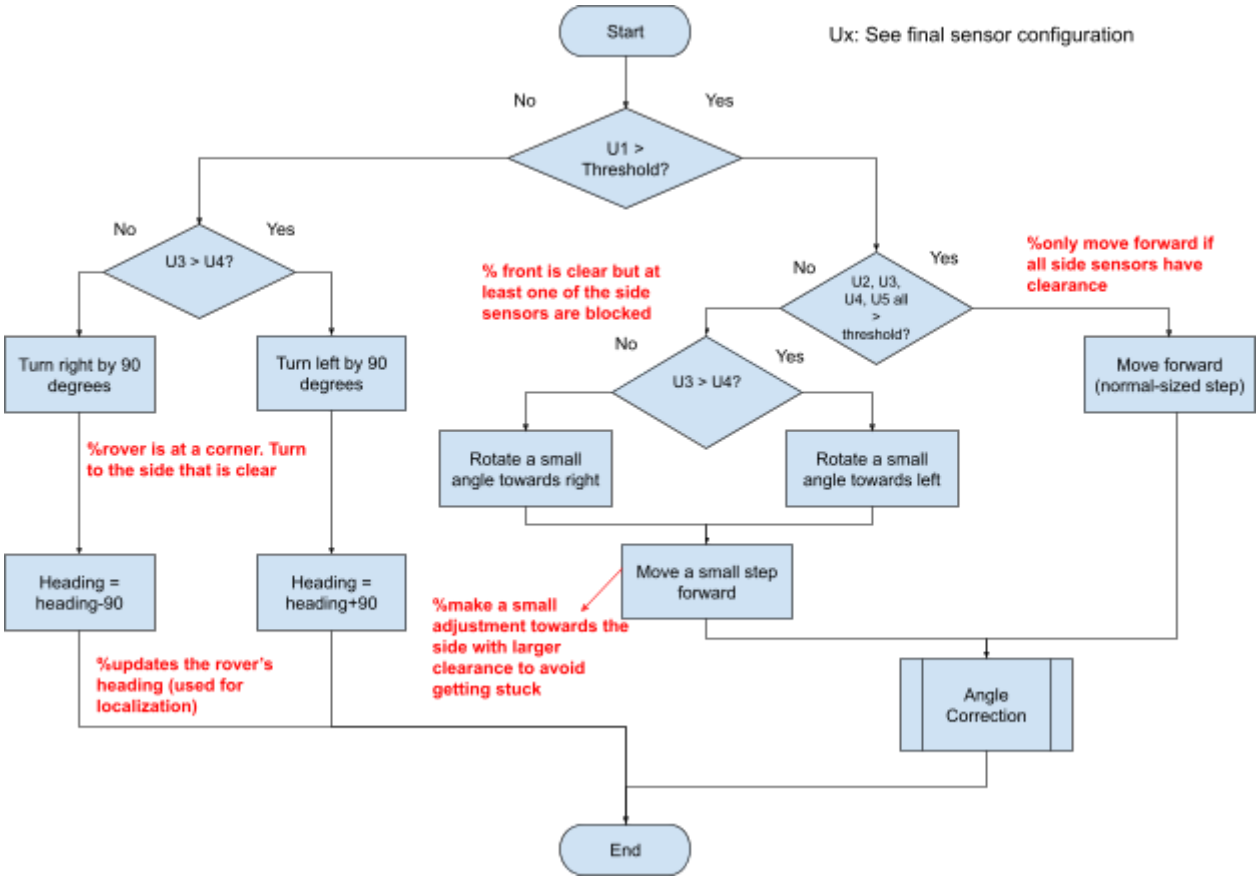

Figure 5. Obstacle Avoidance Logic Flowchart

2.2. Localization & Navigation Strategy

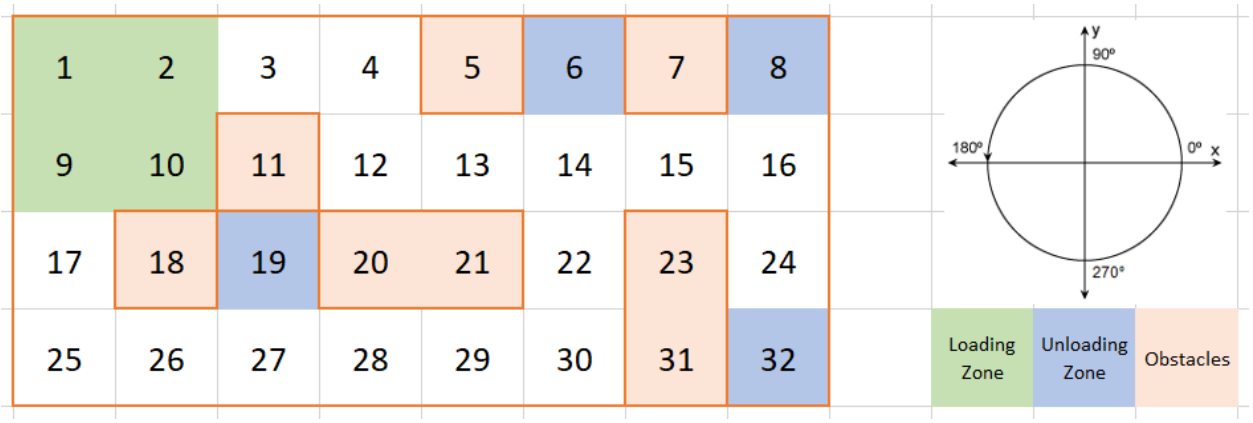

Figure 6. Numbering Convention of the Maze

To localize the rover, the team used an IR sensor to recognize the colour tiles and implemented a modified version of the sample code. Whenever a measurement was taken, it was compared with the actual colour tile map. Then, the code updated the localization probability matrix. Initially, there would be various locations that matched the sequence of the tiles. However, as the rover travelled, the chance of getting the specific sequence from the measured tiles decreased. With enough measurements, there would only be one location that had a high probability (Figure 7).

By trial and error, the team determined that a probability of 10% was a reasonable threshold value for confirmation of successful localization. If the threshold value was higher, it would take too long before the rover started travelling toward the loading zone. If the threshold value was lower, the localization would lack accuracy.

Once the highest probability within the matrix reached 10%, the rover would begin using the localization data for travelling to the loading or drop-off zone. The index of the highest probability in the probability matrix was converted to a block number. Then, using the block number, the rover extracted the destination heading from a predefined array to correctly navigate to either the loading or drop-off zone. The predefined array consisted of 32 elements that corresponded to each block. Each element had two headings: one for the robot to navigate to the loading zone, and another to the drop-off zone.

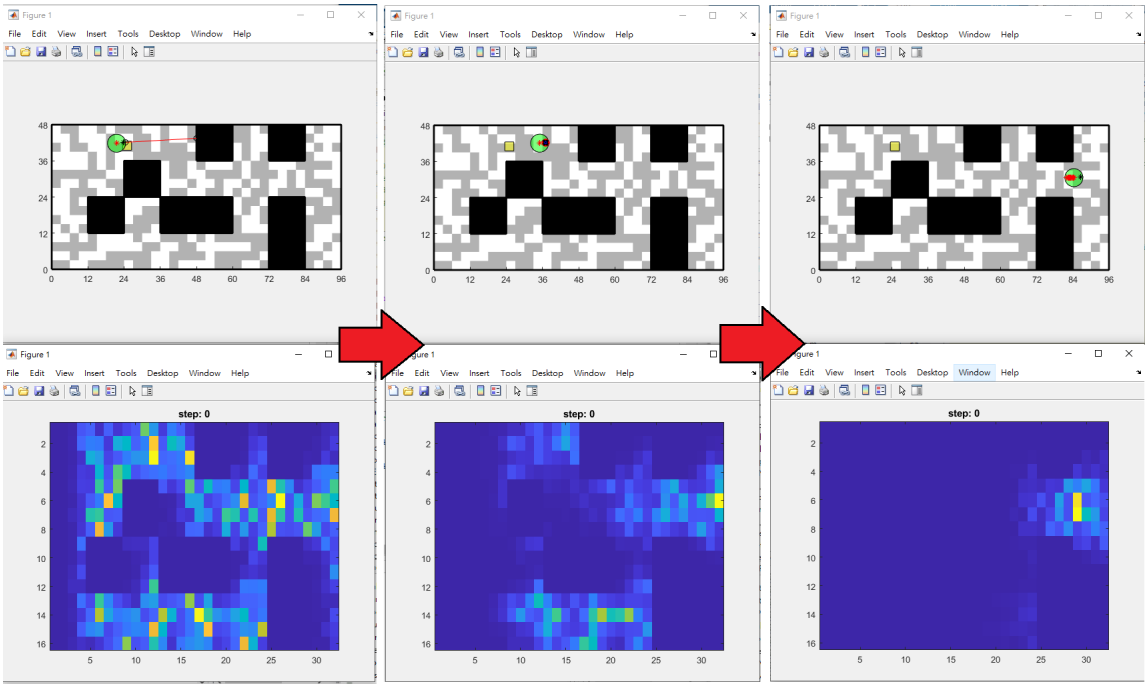

Figure 7: Illustrative Snapshot of the Rover's Actual Position and the Visual Representation of the Localization

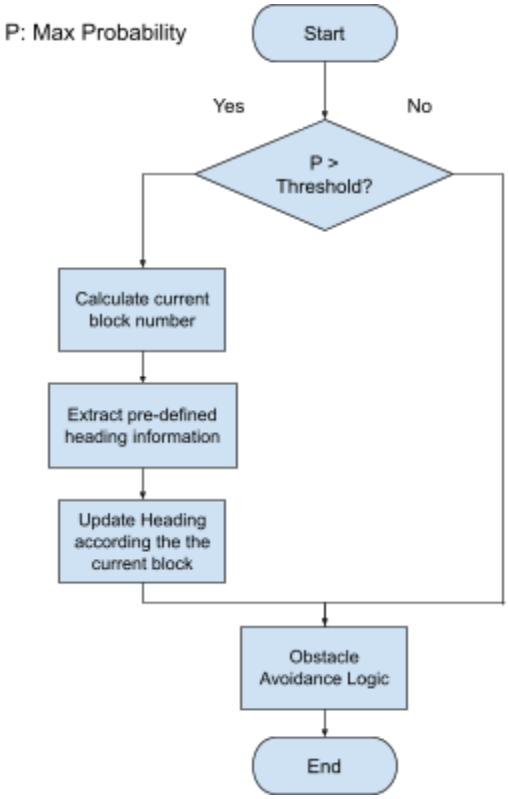

Figure 8: Path-following Logic Flowchart

The team also discovered that in order for the localization algorithm to work well, the heading of the rover needed to match the code setup: If the actual rover heading was different from the algorithm's heading parameter, the probability matrix would shift and update in an incorrect direction whenever the rover moved. Consequently, the localization would not be accurate.

Since localization and fixed-path logics were more integral to completing the overall task, they were developed assuming that the rover would always start in a heading that matched the code. After developing a functional localization strategy, the team implemented a heading correction function. This function used the 4-way clearance at block 14 because it had a unique set of distances. When the rover entered block 14, the ultrasonic sensors would provide a set of distances that match the predefined distance requirement and would update its heading. Once the heading was updated, the rover would turn to the side that was the closest to the loading zone to speed up the testing trial.

During localization, obstacle avoidance logic was still being used. Whenever the robot made a turn, the heading would be updated by adding or subtracting 90° (Figure 9); when travelling along a corridor, the angle correction function ensured travelling straight, thus there were no accidental turns that distorted the heading again.

One last problem was that the highest probability fluctuated around the threshold in rare instances. For the few steps with their probability dropping below 10%, the rover could still rely on obstacle avoidance to travel toward the destination until the probability reached 10% again. This was because the team observed that the obstacle avoidance logic produced a very similar set of headings compared to the pre-defined desired headings once the highest probability first reached 10%.

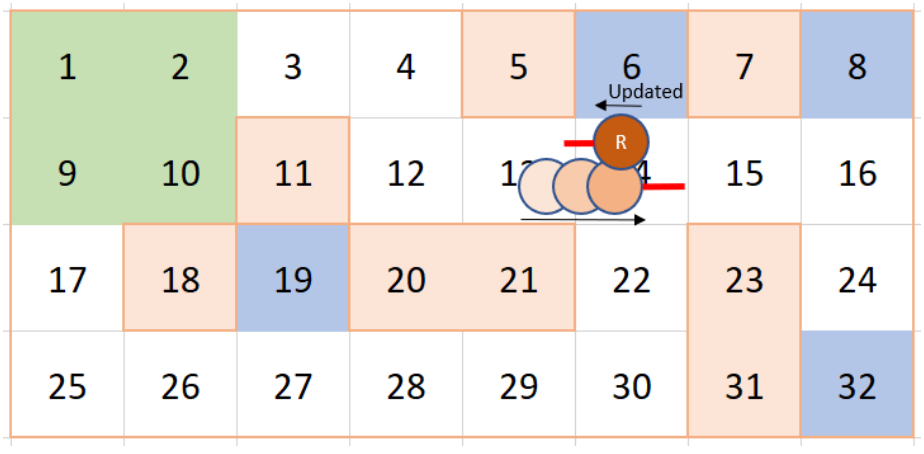

Figure 9. Change of Heading at 4-Way Crossing

For navigating to the loading zone, once the rover identified that it was in either block 3 or 17, it would stop when it was at the edge of the loading zone (i.e. the exterior perimeter), displayed a message to indicate arrival, and began locating and collecting the load. For navigating to the unloading zone, the rover would head into the unloading zone and stop with just enough room to place the load down.

2.3. Block Delivery Strategy

The block pick-up mechanism used on this rover was a 3D-printed mechanical gripper which was controlled by two SG90 servo motors: one controlled the elevation of the gripper which connected the gripper to the chassis and would be referred to as the joint, and the other controlled the opening/ closing of the jaw.

During the initial stages of the rover's operation, before it reached the loading zone, the gripper would be closed and raised to 90°. This design was to allow the same navigation strategy after the load was picked up and the rover navigated toward the unloading zone. Moreover, the raised gripper allowed the load-detecting ultrasonic sensor to be interference-free.

The team's first attempt at the load-finding technique was to only utilize shifting since the motor output low drift/error. The first method consisted of having the robot face block 10 from the beginning and "scan" the entire loading zone by shifting into the loading zone either from block 3 or 17. This method was implemented through the simulator and performed well.

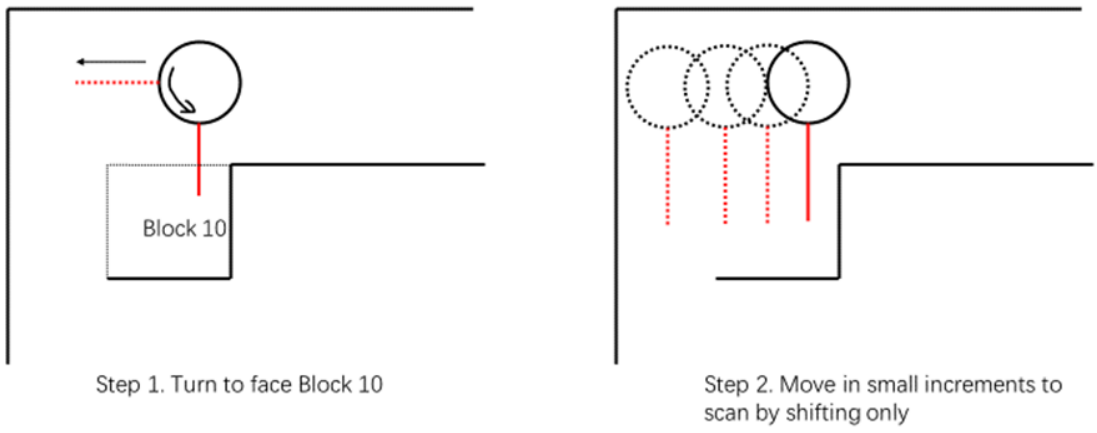

Figure 10. Block-finding Method 1

However, for this technique, improvements were needed due to physical limitations. If the load was placed in the center of blocks 1 or 2 (or 9), the rover would not be able to

detect the load when it shifted in from block 3 (or 7, respectively) due to its size (Figure 8), and it would run over the block.

Design-wise, the load-detection sensor demanded a measuring distance greater than the rated distance of the IR sensor (same location at U2). Therefore, for the final load-detection technique, the team decided to use an ultrasonic sensor instead and implement a more detailed scanning sequence for load-detection.

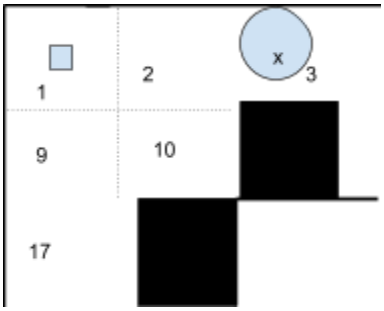

Figure 11. Limitation of Method One

The scanning sequence began after the rover arrived at either block 3 or 17. The rover would activate the front-facing ultrasonic sensor mounted below the gripper. Then, it would perform an initial scan to check if the block was in its straight line of sight by shifting left and right. If the block was not detected, the rover would perform a 90° turn to face block 10. Then, it would shift inside the loading zone (either block 9 or 2). Once the rover entered the loading zone, it would perform a 90-100° scan in 5° increments.

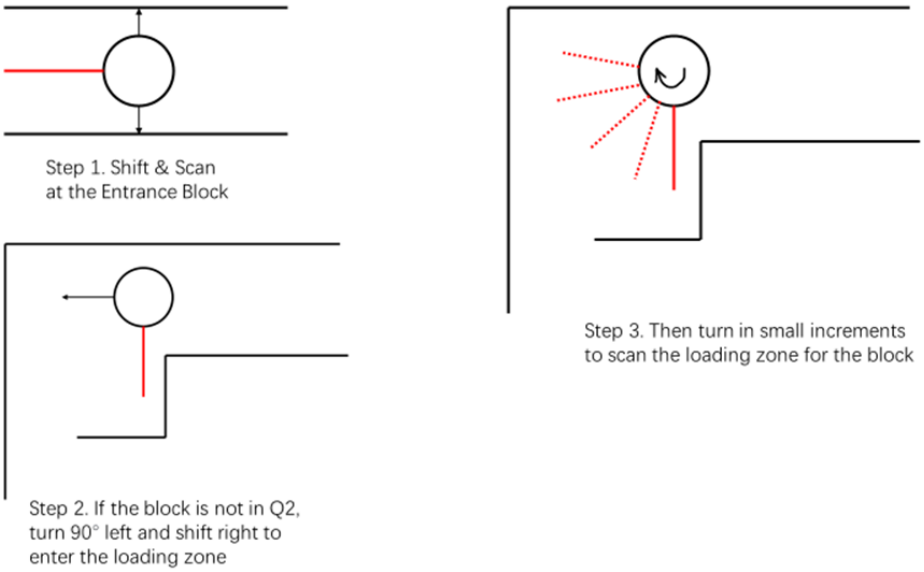

Figure 12. Block-finding Method Two

When the ultrasonic sensor detected the block (i.e. when the bottom sensor reading was less than 50 cm and the absolute difference between the front and bottom sensor was greater than 10 cm), the rover would abort further rotations and drive forward until the load was within the operating range of the gripper. The rover would then open and lower the gripper, place the block between the two clamps, and close and raise it back to 90°. After the load was picked up and stored in the rover, the rover would navigate to the unloading zone using the same technique when it navigated to the loading zone.

2.4. Integration

The complete logic for the project could be summarized using a flowchart in Figure 13. Fundamentally, each individual task (ie. heading initialization, obstacle avoidance, localization, travelling to defined destinations, and load detection & collection) was all packaged into MATLAB functions and called in sequence within an infinite while loop.

The logic can be summarized as the following:

1. Before reaching the 4-way intersection, the rover only executed obstacle avoidance logic.
2. Once the rover reached the 4-way intersection, it executed the heading initialization logic, which corrected the heading and commanded the rover to turn toward the closest side of the loading zone. The probability matrix of the localization was also initialized. After this, the heading initialization would be skipped in subsequent loops.
3. The rover executed localization and obstacle avoidance.
4. If an element in the probability matrix reached 10%, the rover would begin travelling toward the loading zone. If the highest probability subsequently dropped below 10%, path-following logic would be temporarily bypassed until the highest probability reached 10% again (section 2.2).
5. Once the rover reached the loading zone (either from block 3 or 17), path-following and obstacle avoidance logic would be bypassed during load detention and collection.
6. The rover would execute load detection and collection. Upon completion, this function would be bypassed in future loops.
7. The rover executed path-following and obstacle avoidance logic while travelling toward the unloading zone.
8. Once it reached the predefined unloading location, the load was gently placed on the floor and the entire code terminated.

* Obstacle avoidance and localization were mostly running unless otherwise specified.

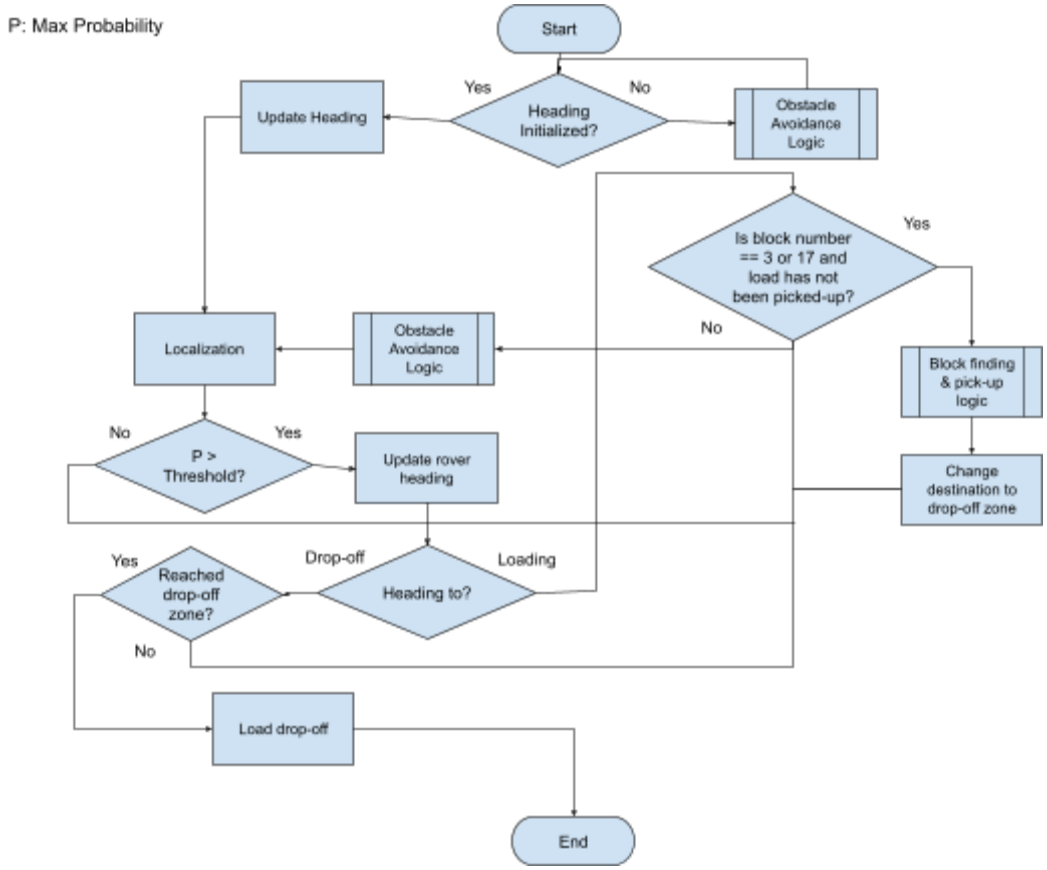

Figure 13. Overall Integration Logic Flowchart

Finally, it is worth noting that the team did not fully integrate the flowchart. Specifically, the team did not have time to incorporate steps 6 and 8 with the rest of the operations. For Milestone 3, steps 6 and 8 were demonstrated individually; for the actual trial, after the rover reached the loading zone, it directly proceeded to execute travelling to the unloading zone. This will be discussed further in section 3.

3. Final Results

3.1. Obstacle Avoidance

The overall performance of the rover's obstacle avoidance met the requirement: the rover did not need any human assistance to travel through the maze and it did not collide with the maze walls. The rover was also able to make minor adjustments to its heading in order to maintain a rather straight path. However, the trade-off for accuracy was speed. The speed of the rover was slow, and the rover was unable to complete all tasks within the time limit.

3.2. Localization

Localization was tested in Trial 1 of Milestone 3. As shown below in Figure 14, the rover started at block 30 with a heading at about 170° and followed the path as sketched with the red line to the 4-way crossing, where it successfully updated its heading. The rover then adjusted its heading to 180° and moved toward the loading zone. Since the team was unable to integrate the load-detection and collection functions, the team set the arrival at block 1 as a trigger to print "I am at the loading zone!" in the MATLAB command window to indicate that the rover had arrived at the loading zone. The trial terminated after the rover reached the loading zone due to a time constraint of 10 minutes.

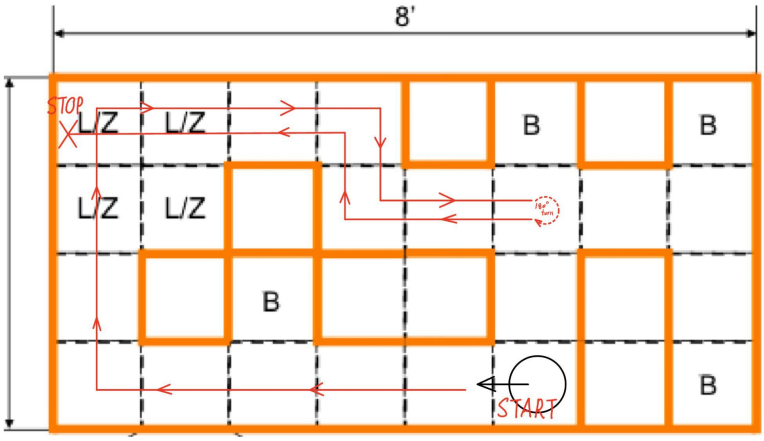

Figure 14. Rover Route for Milestone 3 Trial 1

3.3. Pick-up & Delivery of the Load

The team focused on the load detection and pick-up functions during Trial 2 of Milestone 3. The team skipped the localization section, placed the rover at the exterior of the loading zone (at block 17, figure 6), and activated the block detection algorithm.

The rover first scanned and determined that the load was not in block 9. Then, it entered the loading zone and successfully detected the load, which was located at the center of block 1. Finally, it travelled forward and activated the series of gripper commands to pick up the block.

Overall, the load detection logic was functional as the rover successfully identified the load. However, load pick-up & drop-off failed due to poor gripper control & design. When the rover entered the loading zone, some manual assistance was required to correct inaccurate motor movements and avoid wall collisions. The gripper failed to pick up the load due to jaw misalignment and a lack of torque at the joint. Specifically, as the rover moved forward to the load, motor inaccuracies caused drifting with respect to the load. The gripper lowered to roughly the correct position but the team had to manually place the load in the center of the jaw to secure it. For the load drop-off, the gripper fell off the rover due to a weak connection between the joint servo and the gripper base. Despite this, the gripper was able to open and close at the appropriate time according to MATLAB commands. Lastly, the team did not demonstrate delivering the load to the drop-off location due to time constraints. Delivery was illustrated in the full-run video that demonstrated the robot traversing to both the loading and unloading zones using localization.

3.4. Integration

The team treated the gripper design as an independent section, and several issues, which were identified very late in the project, could not be fixed: While gripper testing succeeded on its own, the team did not account for the distribution of battery power and gripper mounting mechanism. This led to insufficient torque for load pickup and a weak connection between the gripper and the rover.

The team experienced difficulties integrating all the MATLAB code to match the full sequence of the flowchart (figure 13). Specifically, there was confusion in compiling various sections due to unorganized version control. For instance, because the use of functions and documentaries was not stressed at the beginning, various versions of obstacle avoidance code existed, which caused confusion and required much time to troubleshoot. Consequently, the team could not integrate load detection and pickup into the final version of the code.

Overall, the integration of various components of this project was considered poor. Several means to improve integration are discussed in section 4.

4. Discussion

There were many areas of improvement, but some of the major challenges the team encountered were the design of the gripper and code integration of the entire operation.

As discussed, the gripper could have been better designed. The team tested the gripper the day before Milestone 3, thus, unable to adjust and iterate an improved model. At the beginning of the project, two SG90 servo motors were given for gripper operation. However, the team did not consider the power supply for these servo motors until they were installed. Because the team used a single 12V battery to power all motors, Arduino boards, and sensors, the power that the servo motors drew were insufficient to exert a strong grip on the load and lift it.

Initially, the gripper was installed using a press fit, as shown below in Figure 15. However, due to the low precision of 3D printing, the hole was not properly fitted and it enlarged after a few uses. Even after the team attached the gripper using the fitting provided by the factory, the required holding force of the load exceeded the fittings, hence, it fell off the joint motor. A more reliable connection method would be designing a plastic gear that has a slot to hold a smaller metallic gear. Since the plastic gear had larger teeth, a smaller metallic gear would be required to fully transmit the torque from the servo motor as well as keep the gripper firmly attached to the motor.

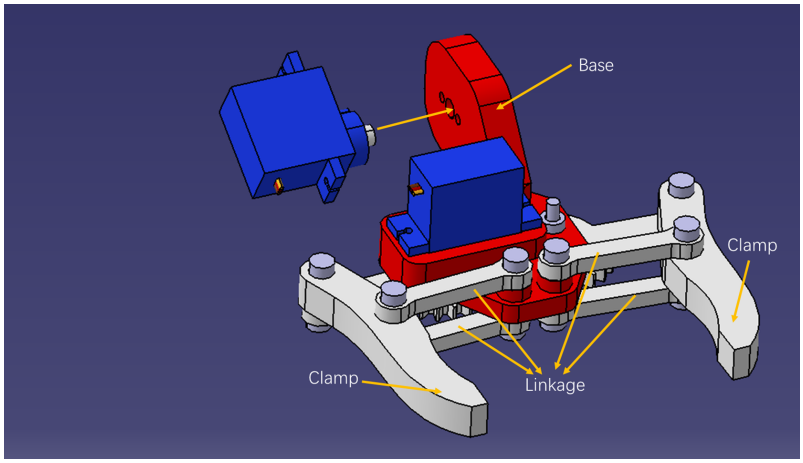

Figure 15. CAD Drawing of the Gripper

Moreover, the size of the jaw was too small. When designing the CAD model of the gripper, the distance between the two clamps of the gripper was 2.64 inches. However, the built-up errors during the rover's movement and inaccuracies of the sensors left very little clearance between the load and the clamps.

If the team were to improve the design of the gripper, the first thing would be to increase the size of the gripper to allow more tolerance for error. Moreover, it would be

ideal to have an individual power supply for the servo motors, so that it can be guaranteed that the servos would be able to provide sufficient torque for load pick-up. Lastly, the team should allocate more time to the gripper in the event that a re-design is required.

The team also had to rebuild the rover after a challenging development for Milestone 1. The first rover had a diameter of 9 inches so the sensors were too close to the wall. This resulted in the sensors' frequent incorrect distance measurements, which directed the rover into the maze wall. The team barely made obstacle avoidance work by moving the sensor as inward as possible. The team learnt from this experience and re-designed the rover entirely.

After Milestone 1, the re-designed rover was smaller, with a diameter of 6.5 in for the top sensor layer and 7.5 in for the bottom motor layer, and performed obstacle avoidance much better. The logic and programming behind obstacle avoidance allowed the rover to navigate within the maze collision-free to any obstacle and follow rather straight paths. The newly improved movement algorithm also allowed the sample localization code to work properly and output very accurate predictions of its actual location.

For the coding aspect, the final code integration took more time than expected due to inefficient code organization. Overall, tools like GitHub could help with the merging of codes for this project. Also, better communication could have given the team an upper hand in realizing the inconsistency early on. In addition, the delay between each rover movement was extremely long compared to SimMer, and this caused the Milestone 3 trials to run over the time limit. If given more time, the team would investigate the causes and hopefully resolve this incompetency.

There were also areas deserving praise, such as the obstacle avoidance and implementation of the Bluetooth module after Milestone 1.

Reviewing our RFP, the final obstacle avoidance logic achieved most of the related functions and objectives, namely navigating through a walled maze without collision and minimizing human interactions during the task execution. The angle correction logic also facilitated the success of localization and travelling to designated locations.
Lastly, the implementation of Bluetooth was a significant achievement because it greatly simplified rover testing and troubleshooting. The Bluetooth connection allowed the team to avoid conflicting with other teams' USB cables, which was convenient because the team could continue testing even when multiple teams crowded the maze area. Also, the team could simultaneously test MATLAB and Arduino codes without changing the

COM ports, which was beneficial as the team could more easily troubleshoot sensor readings during Milestone 2.

Overall, the team members had increased their overall capacity to stress and work under pressure. The team also realized the importance of time management to alleviate the workload in more manageable terms. Moreover, the team gained the ability to predict potential problems and accounted for them during the design phase. On the technical side, the team gained a deeper understanding of mechatronic components, including power protection, Bluetooth connection, wire management, sensor allocation, CAD design, and motor control. Finally, each member learned the importance of version control in coding.