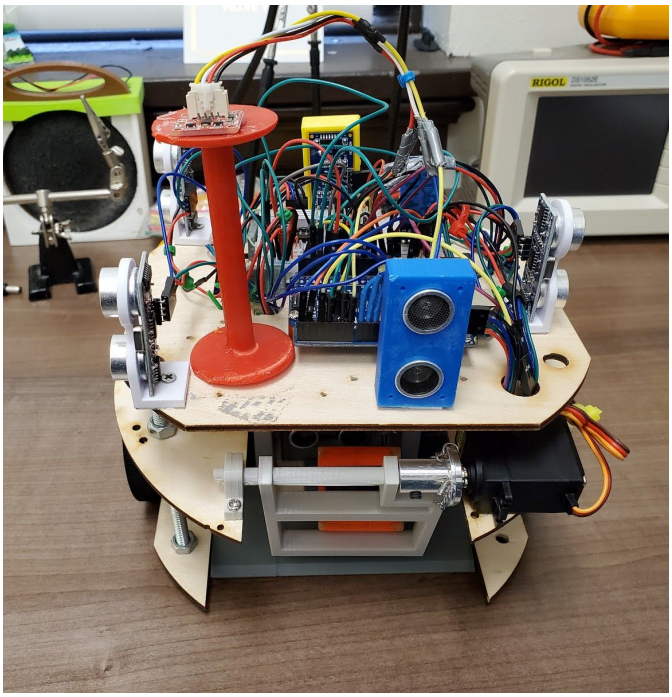
Final Report

MIE444 Project

Catherine Kucaba - 1003278026
Maximilian Glidden - 1002277396
Savo Bajic - 1003051485

12/09/2020



Executive Summary

The goal of this project was to design and program a robotic system that is required to autonomously localize itself in a known maze layout and navigate it regardless of starting position or orientation, as it carries out a specific set of tasks whilst avoiding collisions and obstacles. The maze was walled and had a randomized checkered pattern on the floor. Within the maze, the robot was tasked with locating and picking up a small load (i.e., a wooden block) in a loading zone (LZ) and delivering it to a desired drop zone (DZ or Point B). It was required to perform these tasks within eight minutes. A pre-built robot was provided by the teaching team for this stage of the project.

The team successfully completed this task with the provided rover, perfectly, in five and a half minutes. These results can be found in Week 12 Milestone 3 - Trial 1 - Afternoon/recording_6", starting at time 56:55. This was the first successful completion of the full task by any team during the trial runs. At the time of submission, the rover could reliably localise, navigate to and from the target loading zone, and retrieve and deliver the payload. Algorithms for the control of the rover were developed and integrated in MATLAB. Obstacle avoidance and lane-keeping were achieved using an algorithm with adaptive step size, which sped up the rover's movement considerably, a major challenge the team faced in the early development milestones. Localization was achieved largely using the ultrasonic sensors to detect the distance to adjacent walls. The results of this and previous scans would generate a confidence level for a certain location. Once a location reaches a certain confidence proportional to the remainder of the maze, it would consider itself localized and plot a path to the desired target, either the LZ or B, using an A* algorithm. It would try to follow the path, verifying it's progress until the destination was reached. In the LZ the rover would rotate and scan for the block, centering itself before approaching and grabbing it. It would then realign and relocalize itself in the maze before going to the DZ and dropping the block.

The overall performance of the rover was consistently impressive with minimal collisions, fast traversal of the maze environment, and reliable localization. The team felt there was room for further improvement with the accuracy of the block detection technique, which could be further refined. The speed of the rover in the maze was another target for continuous improvement, and with further tuning, even faster traversal times are achievable.

This project gave the team a better understanding of the design and integration of mechatronic systems, and an appreciation for the challenges inherent in mobile robotics.

Detailed Rover Control Strategy

The rover was controlled using a MATLAB script executed on a computer that communicated with the rover over Bluetooth connection. The rover would execute commands in the order it received them in (first in, first out) one by one as they were completed. The team largely relied on simulations and limited real testing of the rover due to COVID-19.

Obstacle Avoidance

Obstacle avoidance forms the basis for the rover's movement through the maze, preventing collisions with the walls, which allows for localization and navigation. Initially, the team's obstacle avoidance routines made up the majority of the main operating loop. As the team progressed through the project milestones, it became more difficult to integrate obstacle avoidance at a high level without increasing the overall complexity of the rover algorithms. The team made the decision to integrate the bulk of the algorithm for obstacle avoidance directly into forward movement commands, which simplified overall integration. This approach was successful, because the robot risked collision only during movements and rotations. With proper choice of acceptable clearances to surrounding walls, the risk of the rover colliding during a rotation could be mitigated. The obstacle avoidance algorithm served to both reduce risk of collision when moving forward, and achieve the clearances necessary for turning in the maze. The flow of the obstacle avoidance algorithm is provided below.

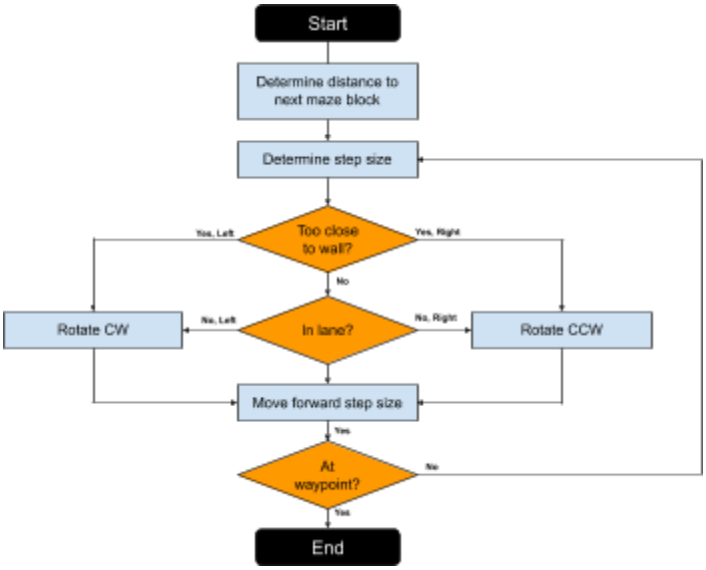


Figure 1: Forward Movement - Obstacle Avoidance Algorithm Flow

Upon startup, the rover initializes a minimum acceptable clearance taken from the edge of the robot footprint to the wall in any direction, as well as a maximum and minimum step size, and a lane width. This minimum clearance is used throughout the navigation process to guide obstacle avoidance. At the beginning of each operating loop, the rover samples each of the ultrasonic sensors sequentially (akin to a low-resolution lidar) and subtracts the fixed distance between each ultrasonic sensor and the corresponding edge of the rover footprint from the raw sensor output. This gives an approximate clearance value between the robot and the wall in each direction. After performing localization and navigation processing, the last action performed before completing the loop is forward movement. A step size for forward motion is determined adaptively as:

$$stepSize = K * \sqrt{forwardClearance}\ ,\ forwardClearance > 12"$$
$$else\ stepSize = 4"$$

where K is a user-set gain factor (default of 1.5 after real-world tuning). This step size is compared against the remaining clearance in the forward direction, accounting for the minimum allowable clearance with the wall. The minimum of these values is taken as the step size. This strategy for determining step size allows the rover to move in larger steps when there is ample space in front of it. After determining a step size, the avoidance algorithm checks the clearances from the left and right ultrasonic sensors. The algorithm tries to keep the rover centred on a predefined “lane” in the middle of the left and right walls. If the rover is closer than the minimum acceptable clearance on either side, it takes evasive action to place the centroid of the rover approximately 6” from the nearest wall (nominally centered on the lane). An angle of rotation is determined as:

$$\theta = arcsin(\frac{6-(R+clearance)}{stepSize})$$

Where *R* is the radius of the rover footprint. The numerator of the above expression can be thought of as the horizontal offset between the nominal centre of the lane and the rover centroid. Direction of rotation is determined by which sensor (left or right) detects a clearance less than the minimum threshold. Rotating the rover by θ, and advancing by the previously determined step size should place the rover’s centroid roughly in the middle of the lane, after which the rover re-aligns itself with the wall of the maze using the two right-side ultrasonic sensors. The rover attempts a similar centering technique when it detects that it has left the lane, by averaging the difference between the left and right clearance, and then generating an angle. This centering process, combined with the adaptive step size, ensured the rover had adequate space to rotate when necessary. Rover rotation for the purpose of navigating the maze was kept

separate from the obstacle avoidance algorithm, and instead integrated into the localization and navigation modules.

The obstacle avoidance algorithm was developed primarily in the provided *SimMeR* Matlab simulator. The differences between real-world and simulator operation presented a major challenge for the team to overcome. One part of this challenge was the level of uncertainty in the accuracy of sensor readings and movements in simulation. Prior to the first real-world trial, the team lacked information on the accuracy of rover movement and rotation. The simulator allowed for random noise to be added to sensor readings and rover movements to account for real-world errors. The default noise ranges for sensing and movement were quite large, ranging from 2-5%. With such large potential for accumulated errors in the simulator, the team initially designed the obstacle avoidance algorithm to move in small, fixed steps. Wall avoidance was achieved with constant sensor scans, large minimum acceptable clearances, and aggressive fixed-angle corrections. While this approach was fast and robust in simulator, it failed spectacularly in initial real-world trials. The aggressive avoidance strategy caused the rover to 'pinball' back and forth between the walls of the maze, over-correcting and moving too close to the opposite wall instead of moving straight ahead. The real-world errors in rover movement and sensing were on the whole much lower than the simulator. This contributed to a larger issue, the rover's speed in the maze. Compared to the speed at which commands could be issued to the rover in simulator, a major bottleneck in real-world performance was the time taken to issue commands to the rover over bluetooth from the controlling PC. With repeated calls to individual sensors, there was significant downtime between each iteration of the operating loop (~20 seconds), after which the rover would make a 4 inch step. In initial trials using this algorithm, the rover failed to travel more than a third of the required distance.

In redesigning the obstacle avoidance algorithm after initial real-world trials, a key realisation was that the real-world errors in rover movement were significantly lower than previously assumed. The team re-tuned the simulator with this in mind, reducing the amount of simulated noise by an order of magnitude. This also drove a complete rethink of the avoidance and movement strategy. Instead of moving in small fixed steps, the team implemented adaptive step sizes, which allowed the rover to move larger distances per step when it had the room to do so and move more precisely in corners and other critical areas. Wall avoidance was also changed to be less aggressive, moving the rover forward at a less drastic angle and centering it in the lane using the adaptive step size. This eliminated the issues with 'pinballing' present in initial trials. Finally, the algorithm was changed to reduce the number of commands sent to the rover per step. This resulted in a rover that moved forward quickly with less downtime, and

with more accuracy; achieving the required distance for the first project milestone. Further tuning of the step size and avoidance algorithms for later milestones further improved the rover's traversal speed.

A new challenge that arose with the improved avoidance algorithm was the effect of maze edge cases on the centering of the rover. In select locations in the maze where left and right walls were not equidistant from the centre path (the loading zone, 4-way intersection), attempting to use the centering algorithm would risk collision as the rover attempted to centre to a lane that did not actually exist. To mitigate this issue, the team added logic to check for these cases prior to a centering action. When such cases were detected, the rover avoided centering at the current step, and instead centered once the location causing the edge case was cleared.

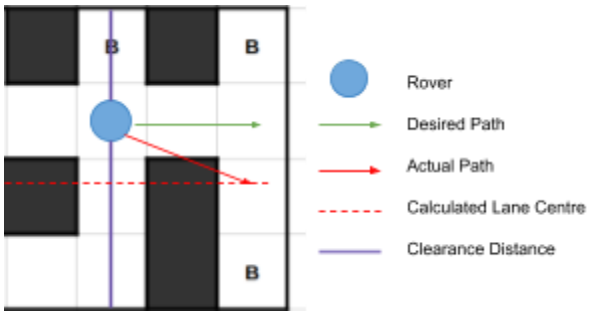


Figure 2: Visualization of Collision due to Edge Case When Centering

Localization and Navigation Strategy

In the rover's hierarchy of needs, localization is the level above obstacle avoidance. Above localization is navigation, as successful navigation relies on proper localization. These were calculated using 12" by 12" squares to describe the maze, and assuming the rover would be roughly centered in the center of any such square when the calculation is executed.

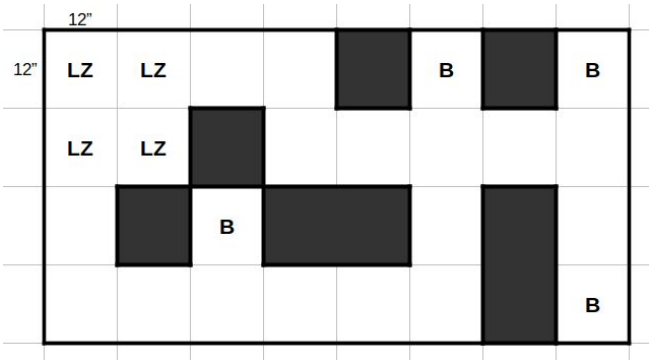


Figure 3: Maze Layout (Top left square is (1, 1), bottom right is (4, 8)) (A heading of 1 points upwards and increases going around counter-clockwise)

Above is a view of the maze the rover had to navigate, broken down into the grid of 12”x12” squares (also referred to as cells or nodes). In the top left is the lift zone (LZ) where the rover must pick up the block from, the squares marked with “B”s are potential drop zones where the rover may need to deposit the block (the specific one is selected prior to the trial).

Localization

Localization was determined using the likelihood of the rover being in a given position based on sensor readings after travelling through the maze. Once the position with the highest likelihood surpassed twice the likelihood of the second position the rover was declared localized. The use of a relative threshold between the highest squares was selected because the coarseness of the grid used in calculations can not tolerate multiple similarly probable locations and be reasonably accurate to reality. If two adjacent nodes share a similar and high value (e.g. 30% likelihood), there are two possible squares the rover could be in, each with different paths to take to the destination. This is not an issue with finer localization grids (such as 3”x3” squares) employed by other groups where adjacent nodes can share similarly high confidences and the path finding would still be valid regardless of this small error in describing the rover’s true position. Below are two figures showing the path found for two adjacent nodes to the “B” square using different grid sizes. With the finer grid (3”x3”) these two nodes would both tell the rover to move right, however with the 12”x12” grid, they disagree.

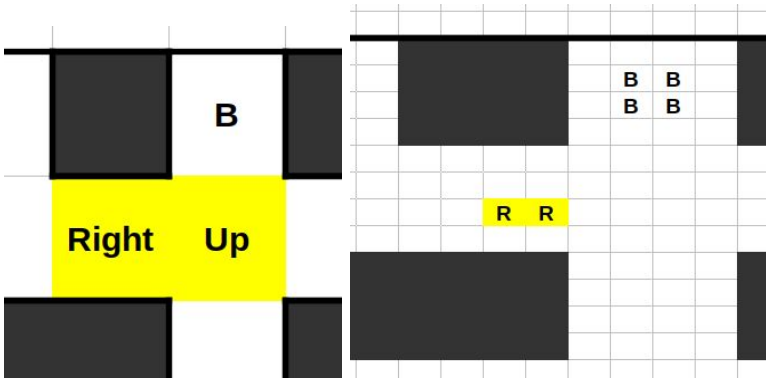


Figure 4: Demonstration of path finding from adjacent nodes using different grid sizes

There were three sources that were combined to determine the likelihood of the rover being at any given position in the maze:

- The walls bordering the square the rover is currently in
- The distance to the first wall in every direction of the rover
- A shifted copy of the previous confidences across the maze based on the movement of the rover

All of these were initially based exclusively using the ultrasonic sensors used for obstacle avoidance. There are two reasons for this; first one being that it sped up the rover since the sensors only had to be polled once since the same values could be used for both avoidance and localization (the reason for localizing before moving), and secondly the team did not trust the compass given then troubles they had using it to accomplish milestone 1 using it. The compass was eventually reintroduced for milestone 3 but in a limited role, simply serving to determine the rover's approximate heading in the maze (e.g. "up" or "right") after discussion with other groups convinced the team with their success using this method.

To determine the likely position of the rover using the adjacent walls the rover uses the ultrasonic sensor readings and records on which sides it detects a wall closer than 6" away, 0 for no, 1 for yes. These are recorded in a string going around the rover starting in the front going counter clockwise looking down on the robot. So if there is a wall in front of the rover and to the right "1001" is recorded. This string is then compared to a predefined set of values for each square to see where it matches. The advantage of using a string like this is that the heading of the rover can be inferred by seeing at which rotation of the string (shifting the letters to the right and looping around), it matches the constant array for the maze (defined assuming the rover is facing up when the strings match.

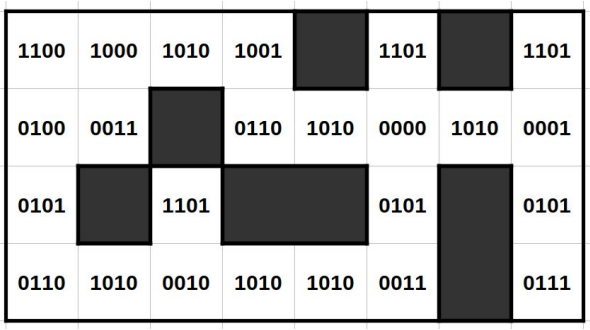


Figure 5: Wall constants through the maze

Using the example of "1001" (wall in front and on the right) there is a match at (1, 4) so the rover would mark that as a potential site. Rotating the string once results in "1100" which has a match at (1, 1) implying the rover may be there, but facing left. Further rotations would reveal other locations at other orientations.

This method of searching was useful because it allowed the heading of the rover to be estimated even without the compass which was vital during the phase the team did not use it. However there are a few cases where the heading would be ambiguous, in the case where there is a wall on the left and right (but not front or behind) of the

rover such as at (4, 2) there are two possible headings, at (2, 6) the heading is completely indeterminant.

The other method used to estimate locations was using the distance (or clearance to walls) for the rover in each direction. This was essentially a development from the wall estimate method. The primary difference between the two is that for clearance testing the distance to the next wall in each direction is divided by 12 and rounded to get a number squares between the rover and that wall. Otherwise the process for determining location and heading is identical between the two using string rotation.

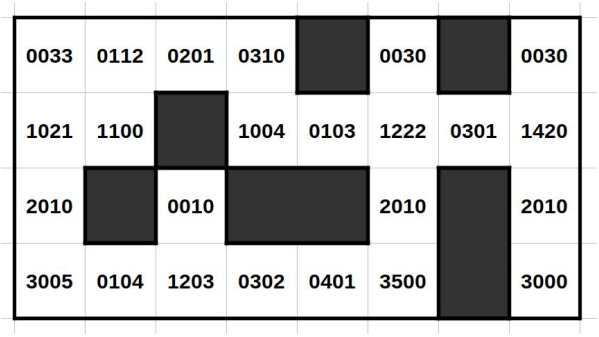


Figure 6: Clearance constants for the maze

The benefit of clearance checking over wall checking is that there are fewer similar cells the rover can think it's in at once. This means that if the sensors work perfectly and the rover is properly aligned, clearance checking should be much more precise and localize faster. Another benefit is that the headings are less ambiguous using the clearance method as there are no squares which have multiple possible headings for a given search string.

The final contribution to localization confidence was the previous confidences translated based on the heading of the rover's last move. Before the compass was used, these headings would vary square to square as there wasn't a definite absolute heading to the maze. The heading estimates from the other two methods would be combined prioritizing the clearance based headings over wall based headings where possible and the probability would be moved accordingly e.g. "left" for the cells where a heading was determined.

This method of translating confidences was very complicated as there were many different cases to consider, for example if there were multiple squares that could potentially translate into a single cell (e.g. (2,5), (3, 6), (2,7), and (1, 6) can all translate to (2, 6)). This method also only worked for nodes where the rover believed it was previously, which made correcting errors difficult. The solution to address this heading

issue and reduce the complexity of the code was to reimplement the compass to determine the rover's absolute heading to the maze and translate all probabilities using that heading. This allowed all confidences to be translated and with much simpler code.

With all three factors calculated, they would get combined through multiplication into a final result that would then be scanned for the highest confidence levels. Below is an example result of the rover in (4, 3) in the simulator and the results. (Blue is low confidence, yellow is maximum confidence).

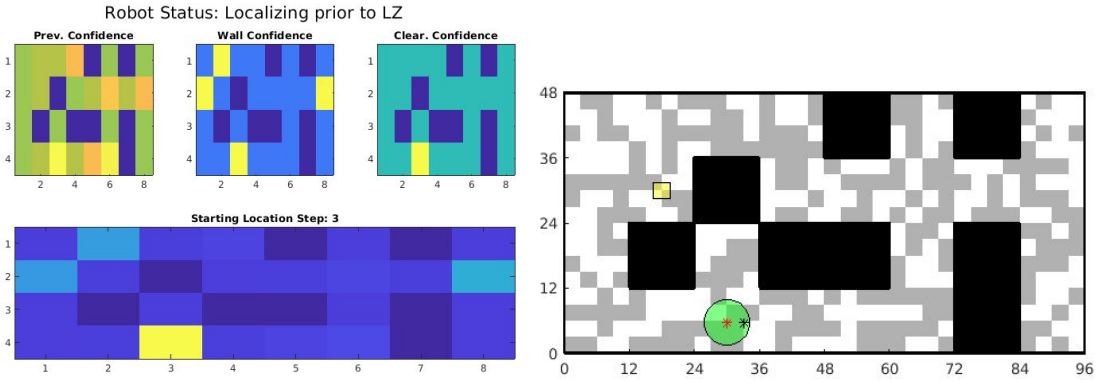


Figure 7: Location confidences (left) based on rover position in simulator (right). The clearance method singles out fewer possible nodes for the rover than the wall method

If the rover was not localized, it would travel in a direction and check its location again until localized. It would first move forward if unobstructed, or else it would then resort to turning left, then right, and, in the worst case, backwards.

One challenge the team faced when going from the simulator to reality was that the ultrasonic sensors were unreliable over large distances. This negatively affected the clearance based localization especially along the bottom between (4, 1) and (4, 6). The team determined the only way to prevent these erroneous readings from severely impacting their localization (as it did in milestone 2) was to decrease the effect of clearance localization on the overall result. This was achieved by taking the square root of it before combining it with the other estimates.

Pathfinding

Once the rover has localized itself, pathfinding occurs using the A* algorithm. Depending on the stage a different target is used. When heading to the drop zone, that coordinate is used as the target. When heading to the LZ, the control algorithm will calculate the route to (2, 1) and (1, 2) to see which resulting route is shorter for the rover. The reason the rover needed to set one of these as the destination instead of (1,

1) which would always yield the shortest route is because the rover needs to stop on entry to the LZ to scan for the block as soon as possible.

A* Pathfinding was selected as it allows the shortest path to be found from any arbitrary point to any other arbitrary point which allows for greater flexibility and less chance for human error than hard coding. The essence of A* pathfinding is that it scans the closest “available” node to the end point. The initial available nodes are the ones adjacent to the starting point. Once scanned a node is moved from the “available” list to the “scanned” list and it’s adjacent nodes are added to the available list. This process continues until the end point is reached. The function would then return the list of nodes the path would take, this was fed into another set of functions to generate a set of functions to generate a list of turns for the rover to take at each node to follow the path.

Pathfinding was one of the few systems that was tested outside of trials or the simulators provided. The team tested it as a separate script that they entered the start and end locations within the maze and then the script would output the node and turn list as well as a figure illustrating the path taken (figure feature was removed when integrated with rover code). The results were compared to hand calculated paths to verify the success of the function.

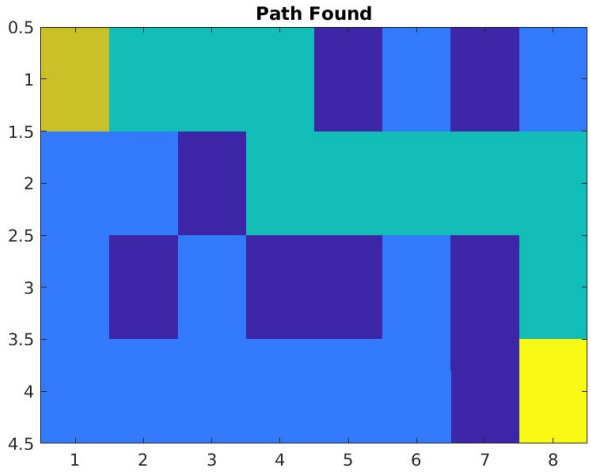


Figure 8: Example figure of path found by the algorithm from (4, 8) to (1, 1) in testing

With the node list and turn list the rover would navigate one square at a time using the obstacle avoidance code. When it would finish arriving at a new square it would verify if the localization agreed that it was in the expected square. If it was, the rover would execute the turn as described in the turn list and repeat. If not, the rover would discard the lists, relocalize and find a new path to the target point. This did not occur frequently in reality but it did occur several times in the simulator that the rover travelled incorrectly or got a sensor reading that misguided it.

Block Delivery Strategy

The block delivery strategy was limited to the act of grabbing or dropping the block. The strategy for grabbing the block was to enter the lift zone, face away from the center of the lift zone, then begin a gradual scan across the lift zone for the block. The block would be detected if the reading for the front obstacle avoidance ultrasonic sensor had a reading exceeding the reading of the obstacle detection ultrasonic sensor that was situated beneath it in the structure.

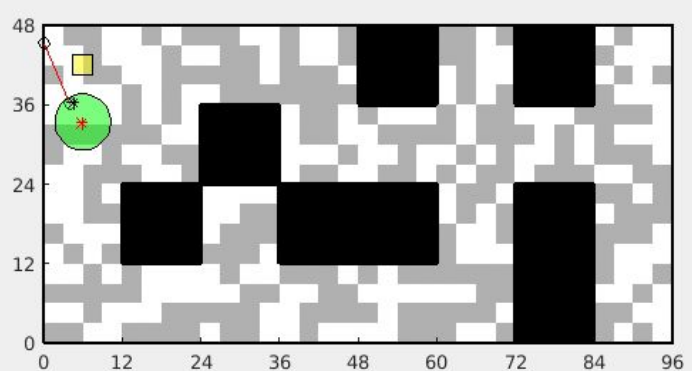


Figure 9: The rover in the simulator beginning its sweep for the block

Once detected the rover would continue its sweep until it failed to detect the block again. It would then aim to center itself on the block by turning to the midway mark between these points and approach the block but leaving enough room to open the grabber without striking the block. The rover would recenter itself on the block, open the grabber, approach the block (hopefully getting it partially up the ramp), and closing the grabber on the block.

At this point the rover would most likely not be aligned with either maze axis. To realign itself to the maze the rover could use the compass or ultrasonic sensors. However the team decided that the faster way would be to make use of the high rotational precision of the rover. This was exploited by keeping track of the cumulative rotations the rover performed in the process of grabbing the block and then sending a single command to reverse this cumulative rotation. This was very successful at roughly returning the rover back to its heading as it entered the lift zone, at which point only one or two iterations of sensor alignment would be needed to align to the maze.

With the block secured, the rover would then relocalize as it would at the start of the run, however once localized it would aim to the drop zone set by the user. Once it verifies it is in the drop zone, the rover would ensure there was enough clearance to open the grabber in front of the rover, open it, then retreat backwards a few inches to drop the block and declare it had done so.

One hardware issue the team faced that they did not expect from the rover was that when the grabber was open, the block detecting ultrasonic sensor would pick it up and constantly return 8 inches and not the actual distance in front of it. This was addressed by adjusting the order of instructions to scan before opening the gripper.

Another issue the team faced was mistaking the edge of the obstacles at (2, 3) or (3, 2) as blocks since one ultrasonic would not detect the obstacle while the other would result in a difference that would be mistaken for the block. This was not observed in any simulation but did occur in a trial for milestone 3, fortunately for the team the second subsequent checks worked as intended and the rover recovered the block on the second sweep. The team did not attempt to address this as the run was successful overall.

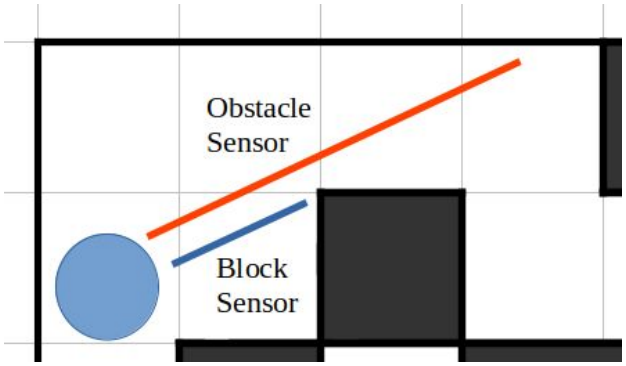


Figure 10: Visualization of ultrasonic sensors on corners

Integration

The overall flow of our control algorithm for the rover is an initial alignment loop for the rover to set itself aligned to one of the axes of the maze. Once aligned, the rover enters the main navigation loop where it will behave differently based on the “state” the rover is in. There are three types of states: localizing, path following, and grabbing the block. This loop will bring the rover to the lift zone (LZ), grab the block, and then go to the drop zone (DZ). Once in the drop zone the rover will exit the navigation loop and execute the block drop off procedure. This flow is summarized in a flowchart (figure 11).

Although there are three types of states, there are actually five unique states present (detailed in table 1). This is because the rover repeats localization and path finding after it grabs the block in the LZ to go to the DZ, other than the difference in destination the behavior of these states is identical. The use of a series “states” within the same loop is advantageous for our algorithm as it allows the rover to easily switch between states to recover from localization errors by changing a single variable. This also carries the advantage of reducing the amount of code needed since states can share code.

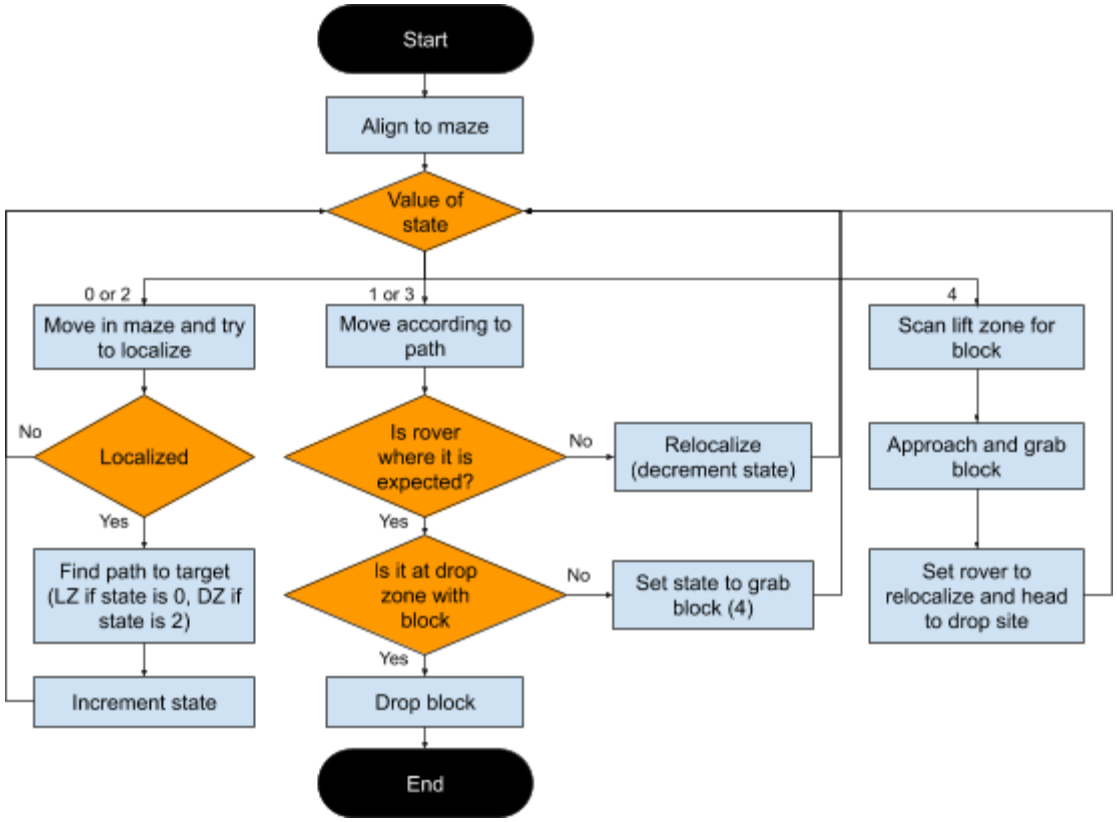


Figure 11: Rover flow diagram

Table 1: Summary of rover navigation states

State	Meaning
0	Localizing prior to picking up the block
1	Path following to lift zone
2	Relocalizing after grabbing the block
3	Path following to drop zone
4	Grabbing block

Final Results

After countless trials and practice runs, the team's rover managed to successfully complete its tasks in the maze in a time of 5 minutes and 20 seconds. These tasks included localizing itself in the maze, navigating to the loading zone, detecting and picking up the load (i.e., a block), and dropping off the load at a programmed drop location, all whilst avoiding collisions and contact with obstacles. This achievement can be seen in the video "Week 12 Milestone 3 - Trial 1 - Afternoon/recording_6", starting at time 56:55. The breakdown of the robot's performance per milestone is discussed below.

Obstacle Avoidance

For Milestone 1, the robot was required to navigate 20 feet (squares) in the maze in 8 minutes, while also avoiding obstacles, such as collisions with the walls. Initially, the team struggled to develop an algorithm for obstacle avoidance that was both reliable and fast. In the practice run after trial 1, the robot managed to travel 7 feet; however, it did so very slowly and inefficiently. In an attempt to increase the speed, the compass readings were utilized. This resulted in more collisions and the rover only travelling 4 feet.

During trial 3, the team's rover successfully navigated 13-14 feet in the maze in 8 minutes without collisions. This can be viewed in the video "Week 10 Milestone 1 - Trial 3 Runs/recording_1" starting at time 02:15:26. This is the furthest distance the rover travelled for Milestone 1 trials and practice runs, in addition to having no collisions, as all or almost all previous runs had at least one collision. Although the criterion of 20 feet was not met, the team received a perfect score for this milestone. Future iterations of this rover could be used to increase the speed to meet the 20 feet requirement, as well as creating a more robust obstacle avoidance code so that increasing speed does not increase collisions.

Localization

For Milestone 2, the robot was required, within 8 minutes, to localize from a random starting position, navigate to the loading zone, and then travel to the drop off location, providing confirmations at each stage. At this point, the rover was moving significantly farther and faster than in Milestone 1. In each run, the robot would localize itself quickly (within a few steps), but also lose localization quickly, especially in the 6 foot section of the maze. This was due to the initial method used for localization, in addition to collisions causing issues.

Trial 2 was the best performance for the team, and can be seen in the video “Week 11 Milestone 2 Trial 2/recording_1”, starting at 02:19:45. In this trial, the robot localized fast, within 1 or 2 steps, and it was better overall at localizing and staying localized than the previous runs. Unfortunately, the rover collided with the wall, causing it to lose localization. It managed to relocalize and successfully navigate to the loading zone. After leaving the loading zone, the rover was unable to navigate to the drop off location due to an error in the pathfinding code. This error was resolved for the next Milestone. Therefore, for Milestone 2, the robot only achieved two out of the three requirements.

Pick-up and Delivery of the Load

For pick-up and delivery in Milestone 3, the robot was required to drive to the load with confirmation, pick it up, place it outside the loading zone, and deliver it within 5 minutes. Initially, during practice runs, the robot would navigate to the loading zone, but would not properly detect the block. When it performed a sweep of the loading zone, corners of the area would be detected instead of the block. In one run, the rover detected the corner as the block and executed the code to grab it. It then navigated to the drop off location and would have successfully dropped off the block if it had it. This issue appeared to be caused by the block ultrasonic sensor differing from the front ultrasonic sensor when facing a wall. During trial 1, the team’s rover successfully detected, picked up, and dropped off the block in a time of 2 minutes and 50 seconds, well within the limit. In addition, the robot did not experience collisions. This can be observed in the video “Week 12 Milestone 3 - Trial 1 - Afternoon/recording_6”, starting at time 56:55.

Integration

All of the requirements above for obstacle avoidance, localization, and block pick-up and delivery were needed to be performed by the robot in Milestone 3. This includes the robot localizing from a random starting position, arriving at the loading zone and detecting the load, picking it up, and delivering it to the drop off location, with confirmation being provided at each stage. This is in addition to being completed in 8 minutes and the robot not contacting obstacles throughout the run. The team’s rover successfully completed the tasks in the maze in a time of 5 minutes and 20 seconds. This can be seen in the video “Week 12 Milestone 3 - Trial 1 - Afternoon/recording_6”, starting at 56:55. The robot localized itself in 2 steps and managed to stay localized as it moved to the loading zone. When it arrived, it performed a sweep of the area and detected the block, aligning itself perfectly to grab it. Once the block was picked up, the robot moved in the loading zone to relocalize itself and then navigated to the drop zone, where it successfully delivered the block. During the entire run, the robot had no collisions.

Discussion

Reviewing the hardware design of the rover, there is one feature that was immeasurably helpful to preparing the control algorithm: the drive system. Using stepper motors for a differential drive led to motion that was accurate, precise, and repeatable. This allowed the team to be less concerned about veering off course with large steps or that adjustments would not be executed correctly when needed. This enabled the team to spend less time tuning obstacle avoidance and more time working on and testing other aspects of the system, especially during the limited real testing runs.

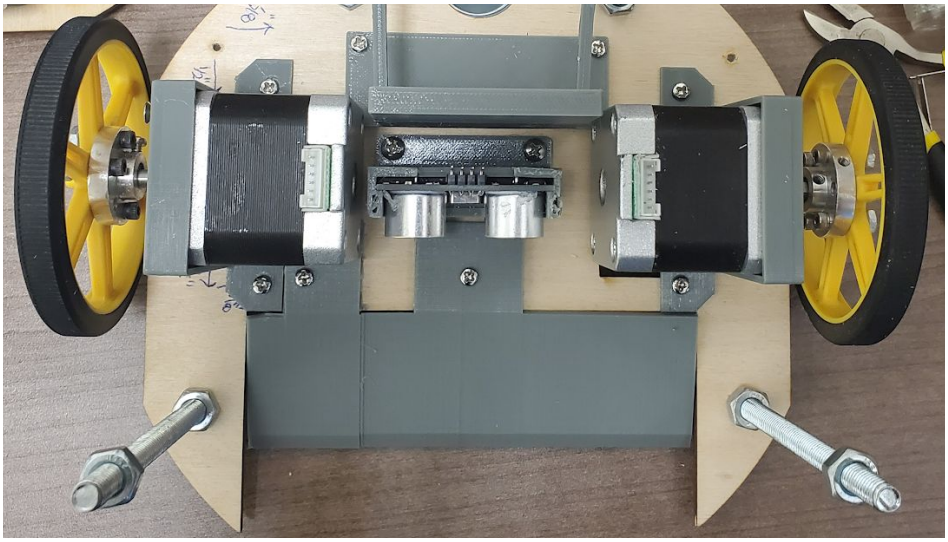


Figure 12: Overhead view of the robot base. The motors and block detector were mounted here and the block would be picked up using the ramp.

The rover did have some weaknesses in the team's opinion. The first issue the team had with the rover was the compass sensor not behaving as expected. It was not linear in the trial demonstrations and was observed to not have a one-to-one relationship with the values it provided and its orientation in the world. During the "ua" demo, the sensor values go from 65º to 110º, then back to 70º while the rover was rotated less than 180º. This means there were multiple possible real headings for sensor values in the range of 70º to 110º. The team only regained some faith in the compass after discussion with other teams and reviewing their trials to verify that the compass did indeed behave better in the maze itself, and reintegrated it into the code in a limited capacity as mentioned in the "Localization" section. The team would suggest a different compass be used in the future, either a direct replacement (should the unit on the rover be simply damaged) or a replacement with a different compass module.

The team also had feedback regarding the ultrasonic sensors, in that the addition of more would have been helpful for block detection. This is primarily to help detect the block if its face is not parallel to the main detector by mounting additional ones in the vacant spaces on the base (shown in Figure 12) at an angle relative to the centre ultrasonic sensor. Additionally these could help prevent false readings of the corner, as the team encountered, by ensuring most of the sensors on the bottom detected the block. The modification to mount the ultrasonic sensors vertically for obstacle avoidance was a nice trick that the team will remember going forward. It allowed for better readings of the surroundings especially around edges, and spreading the two ultrasonic sensors used for measuring alignment to the walls so the difference between them was more pronounced.

Although not possible to implement given the nature of the course during COVID-19, the team learned the importance of splitting computation and decision-making between the main controlling computer and the microcontroller on the rover. There was high latency on any communication over bluetooth (roughly one second) so providing the robot with multiple instructions was time consuming, especially for repeated tasks such as scanning the environment. The power of off loading computation to the rover was shown in the development of the “ua” command during the semester. Initially, each ultrasonic sensor had to be checked individually, taking 5 seconds to check all five sensors for obstacle avoidance once, cutting it down to 1 second to average two readings from all sensors, to eventually marginally over 1 second if more samples were requested using “ua-#” command. If the team does a similar project in the future, they will remember to consider offloading these repeated operations to the microcontroller to improve operation speed.

Reflecting on the code the team created, there were several improvements that were identified. The majority of which are related to dealing with the imperfect reality the rover operated in. Most of these issues specifically relate to block acquisition, which also was the least tested part of the rover control algorithm.

An improvement that could have been implemented in the team’s code for better localization would be to compare the compass heading to the estimated heading for a node. If the real heading matched the calculated heading for the node, it would be marked as a possible location for the rover. If they did not match the result would not be recorded. This would help reduce the redundant matches for wall method and false positives for the clearance method.

The rover didn’t have a system to avoid mistaking the corners of obstacles as blocks nor did it have any method of verifying, let alone handling, if it missed the block,

either during its scan due to the block not being aligned with the rover or it not properly grabbing it. A possible counter to not finding the block at any point would be performing a blind sweep of the entire LZ trying to grab the block in every square. The rover could also be set to check the block was successfully grabbed and stowed using the bottom ultrasonic sensor. However, since the rover successfully completed the task without these measures being implemented, they were never added. The principle of the design and the methods used to detect the root causes to address will be useful tools to takeaway for the team.

Overall, the team will walk away from this project having a better understanding of the systems that compose an autonomous rover, including the extensive mechanical, electrical, and programming design that is required. As was seen in the project, these types of design are intrinsically linked in mechatronics design. The design of the mechanical system will impact how easy it is to code the movements of the rover. The design of the electrical system will impact the way in which the mechanical design is approached in order to properly place each component. During this project, there were many issues with, not only the team's conceptual design, but also the rover that was built and used for testing. This resulted in aspects of the design having to be modified throughout the process, such as adjusting the location of the wheels for better turning, changing the mounting of the ultrasonic sensors for better readings, etc. What this shows is how intricate and iterative mechatronics design is. The team had a small glimpse into how this works and will utilize what was learned moving forward in their careers.

Appendix A: Contribution Table

Contributions graded as; 1 - small amount, 3 - majority, blank for none.

Table 2: Contribution

	Catherine	Maximilian	Savo
Executive Summary	3		
Obstacle Avoidance		3	
Navigation and Localization			3
Block Pickup			3
Integration			3
Results	3		
Discussion	2	2	