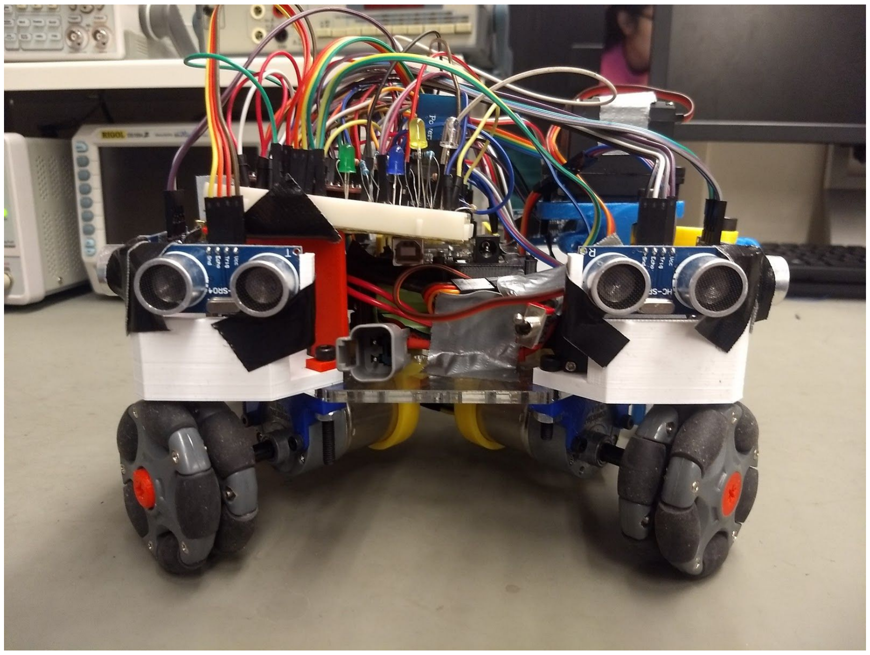
Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

Final Report - Group 5
MIE444 - Mechatronics Principles



Team Members:

Team Member	Student Number
Aldrin Dave Perez	1001525735
Jiayu Hu	1001303939
Anthony Raso	1001531595
Sumant Bagri	1005644313
Rahul Raj	1005644322
Kyle Mccarroll	1002582984

Executive Summary

The goal of this project is to design a robot capable of finding and picking up a block in a maze and deliver it to a predetermined location. To do this, the robot must be capable of avoiding obstacles, localizing to determine its location, and find and pick up a block. Obstacle avoidance is achieved using the distance values obtained from eight ultrasonic sensors, two per side. The robot moves a predetermined block interval, after which it compares the readings from the sensors on the face close to a wall and uses PID control to realign itself moving a set distance from the wall so that walls are avoided when it moves the next block interval. Localization is achieved by comparing the sensor values from each side in its starting position to the sensor value readings eight blocks later. The values determine how many walls are surrounding the robot, and compares them to the maze map to determine its location. It then moves to the loading the zone where the block location is determined by comparing the ultrasonic sensor readings on the front face to the ultrasonic sensor reading behind the gripper. The block is approached with the gripper open and picks it up when in the gripper's range. The robot then moves back into the maze to the unloading zone where the block is dropped.

For the first milestone, the robot avoided all obstacles and worked fine. For the final milestone our runs needed to be pushed back to the second available date as the remaining two functions were not working. On the final date, the first attempt took over the 5 minute limit due to not localizing on the first 8 blocks of movement, while the second attempt was achieved under the time limit. In both cases, it picked up the block, however, on both attempts, walls were grazed twice.

Detailed Rover Design

The overall rover design goals have remained unchanged from the initial proposal, maintaining a roughly square shape, with an 8” x 8” footprint, and a height of approximately 5”. The robot can maneuver easily through the 12” wide corridors with translation-only motion, which the team denoted as “strafing”, but still rotate should it need to. The gripper assembly is mounted on the “front” side of the rover (See Fig. 1), with the assembly inset to prevent contact with the wall.

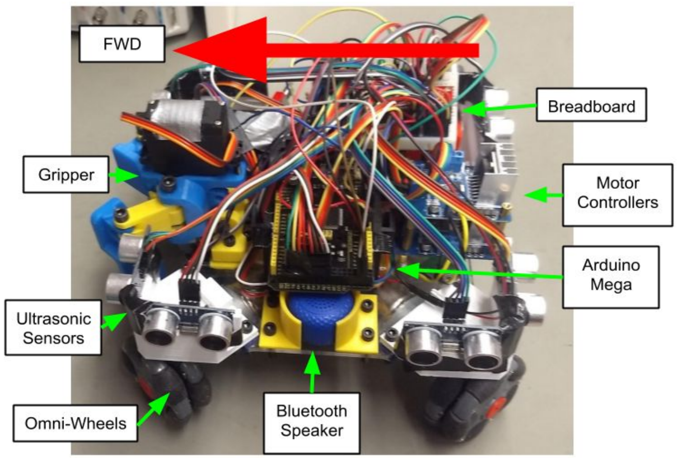


Fig. 1: The final robot design, with the forward face indicated.

The rover uses a laser cut acrylic sheet as a baseplate, with the drive motors mounted below in a cross shape allowing the use of omni-directional wheels (see Fig. 2) and the control electronics mounted on top.

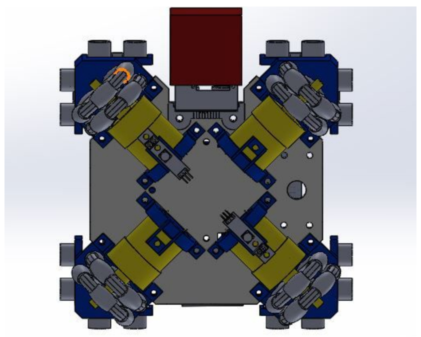


Fig. 2: The bottom view of the CAD model, with the drive motors in yellow.

The robot is controlled by an Arduino Mega because the amount of digital inputs required would have saturated the 14 digital inputs on an Arduino Uno that both the proposal and Milestone 1 rover featured. The Mega is still positioned above the battery using a 3D printed mount to conserve the limited space on the top of the rover. Two dual-channel motor controllers are stacked using brass standoffs, again to conserve space, and mounted to the rear of the rover. This is an improvement to the Milestone 1 design which used a single dual channel motor controller as the team needed to be able to control each of the omni-wheels independently to allow for robot rotation. The rotation ability was added to facilitate both the block search and random starting orientation requirements.

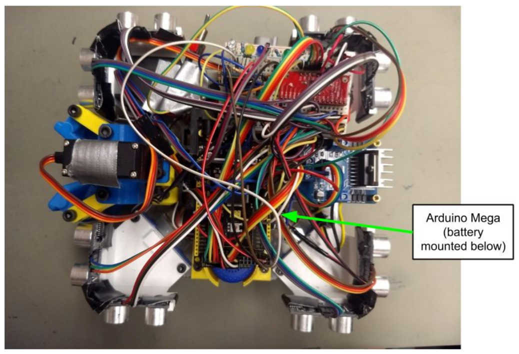


Fig. 3: A top-down view of the rover, highlighting its square shape.

A pair of perpendicular, outward-facing ultrasonic sensors are mounted in a 3D printed structure over each motor as seen in the proposal and Milestone 1 making two ultrasonic sensors on each rover face (See Fig. 4). As the rover moves through a corridor, it will be able to check both sensors on a face to ensure the rover is aligned straight, and far enough from each wall. The initial plan of mounting IR sensors facing downward for localization purposes was abandoned because it was unnecessary as the robot could localize and complete the maze without them.

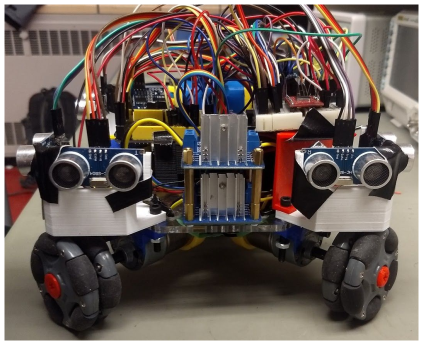


Fig. 4: The ultrasonic sensors are mounted as close to the edge of each face as possible.

The gripper assembly underwent four iterations over the testing period. The initial design used two servo motors, one to tilt the gripper assembly, and one to close the jaws. The original design stored the gripper and the gripped block above the rover to conserve space making it easier for the robot to maneuver although was abandoned due to the lack of space to mount the rotating assembly. There were two more revisions before implementing the final gripper with jaws that extend low enough that the rover can drive forward and close the grippers without having to tilt the gripper downwards. The gripper featured a more compact upward tilt mechanism utilizing a second servo, tilting the gripper about 20 degrees to prevent the block from dragging as the rover moved. The rover used a single ultrasonic sensor mounted below the baseplate to search for the block and know when it was close enough to close the grippers. The assembly was kept very compact, and kept inset to prevent it from colliding with walls.

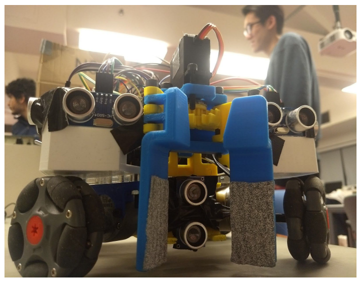


Fig. 5: The gripper assembly.

Maze Solving Strategy

Obstacle Avoidance (Link to PID testing)

During each code loop, the robot reads the value from every ultrasonic sensor and evaluates feasible directions (among 4 cardinal directions) based on the comparison of the smaller sensor value on that face (because there are two ultrasonic sensors on each face) with a desired threshold value. The robot then chooses one of these directions as the next direction of motion and drives the motors accordingly. To correct for any non-idealities in the motion of robot, the following PID corrections were incorporated:

1. Wall closeness:

   If the rover gets too close to a wall, a component is added to the direction which would make it move away and the value is decided based on PID calculations. As soon as the robot is greater than the tuned activation distance, all the PID variables are reset.

2. Robot orientation:

   Since both robot and maze are rectangular, by aligning the robot with one wall, it can be assumed that the robot is aligned with all other walls as well. Thus, only one wall is used for alignment. First the robot checks if a face has a wall available to check against. A PID instance then tries to minimize the difference between two sensor values on that face as long as the wall is present. After the wall is no longer viable, the PID correction is reset.

3. Wall following:

   Even with the previous two factors, the rover had some issues navigating where there was a wall only on one side. The robot would slowly drift towards an open side and would hit any approaching obstacle edges. The third correction simply maintains a distance from a wall which would make the robot move straighter. The correction can be from either the left or right face based on the forward heading, as long as there is a wall present. Similar to the orientation correction, the rover tires to maintain minimum difference between the sensors on the wall-following face.

Localization

The initial, more robust approach was similar to the one provided in class, the major difference being omni-wheel drive. This lead to the rover not needing to rotate for navigation. The discretization and value matching algorithm was also different. For maze solving, a breadth-first search algorithm was used, where orientation of the robot was an immediate output of the localization. As this approach was not a part of our final design for Milestone 2, the details are in the Appendix.

The final, hardcoded approach had the robot localization using the unique blocking/non-blocking states of each maze location as shown in the figure below.

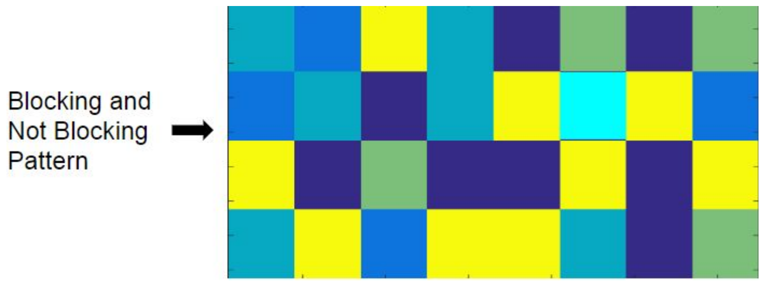


Fig. 6: The blocking pattern of the maze.

Colour at each block represents the blocking status at that robot position. Therefore, with just sensor values, the robot can evaluate the colour of its current block. The core idea behind the team's version of hardcoded localization is to read a sequence of colours generated as the bot moves in discrete steps of one block length. Using a large enough sequence, with final design using 8, the robot can map its sequence to the world map (See Fig. 6) and know its exact current location and its path forward. The robot is hardcoded to only move in the heading of gripper. If this heading is infeasible, it rotates towards a face that it sees as clear.

Block Pickup

After localizing, the robot knows its current location and also its target location. Thus it is able to decide the sequence of motions necessary to move to block pickup. After reaching the entrance to the loading zone, the robot starts a block search sequence, with the gripper open, until the point where the ultrasonic sensor attached behind the gripper detects the block and confirms that it is within the grasp of the gripper. At this point, it closes the gripper which triggers and moves back to the starting point where the block search sequence began. It then moves along a predetermined route to the designated unloading zone, based on which unloading zone the team is assigned. After reaching that position the robot moves a certain distance into the zone, drops the block, and move back out of the zone to confirm drop off.

Final Results

The final version of the robot successfully achieved localization and was able to reach the loading zone to pick up the block and drop it off in one of the four drop-off locations avoiding obstacles in the process.

Obstacle Avoidance (Video Link)

The very first version of the robot was only able to avoid obstacles with a linear and continuous motion as we were using just one motor-controller (so no rotation). This ensured that the robot moved in a straight line, as two opposite wheels were connected together, but was unable to rotate. In the final version we used two motor-controllers to control each motor (without encoders) individually to enable rotation and converted continuous motion to discrete to help localize efficiently. Since we were not using encoders we had to rely on the surroundings to keep the robot aligned and move it in a straight line. This approach was not always reliable as when the robot could not find any surrounding walls (the unique location in the maze) it could not realign itself and would not move in a straight line until it found a wall to correct its position and orientation. This left the robot in a nondeterministic state and it would potentially touch/graze a wall in its subsequent motion. In the other cases the robot was able to accurately correct its position and orientation in each iteration of its motion with help of PID control and thus avoid all obstacles.

Localization (Video Link Trial 1) (Video Link Trial 2)

The robot was localizing using the unique blocking/non-blocking states of locations in the maze as mentioned before. The drawback in this approach was that the robot had to move eight-12 inch-steps in order to obtain a unique “non-headed linear-sequence” (does not have the property of direction and shape) of such states. An incorrect sequence would be obtained if the robot did not move the correct amount in each of the eight steps (if the robot was able to localize in fewer steps chances of generating an erroneous sequence would decrease) or if the sensors produced incorrect readings due to inherent errors (the only solution to this would be to use higher quality sensors). In such a situation the robot would re-start the localization process and thus take more time to localize. This happened during trial 1 of Milestone 2, the robot restarted localization twice before localizing. A solution to this would be to model the sequence as “lego blocks” (Headed-Shaped-Sequence) and then fit those blocks in the maze.

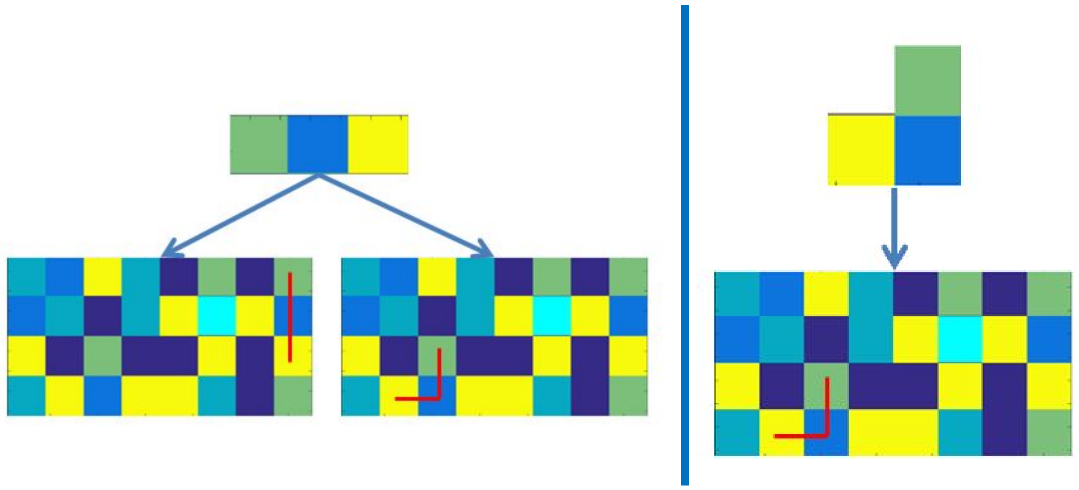


Fig. 7: Linear sequence for localization (left) and “Lego” sequence for localizing (right).

If at any point in the motion the bot realizes that it can fit the “lego block” in more than one position it will move one more step and then check again (localizing with dynamic number of steps). This might drastically reduce the number of steps that the robot has to move (probably even 3 steps as shown above) before it localizes as this would be a one-to-one “sequence to location” mapping eliminating the cases of multiple positions getting mapped to the same “non-headed linear-sequence”.

Block Pickup and Delivery (Video Link)

When the robot reaches the loading zone after localizing it starts a block search sequence which runs until it hits the criteria where it believes that the block is in its grasp. The distance moved in this process is consequently variable. However once the robot “finds” the block it is programmed to move a constant distance to go back to its starting location which may be incorrect if it does not match the distance it moved forward while searching for the block ultimately performing poorly when navigating to the drop-off location. A solution to this would be to check the distance from the wall in front of it after it has picked up the block and then move backward based on that value. Notwithstanding the above, the robot was able to move to the designated unloading point and successfully drop-off the block inside the zone.

Discussion

Changes in Robot Motion and General Design

- Prior to Milestone 1:

  In order to navigate the maze in a continuous manner, the robot was designed to move without the need to rotate. The motion was smooth as no delays were present. Only one motor controller was in use at this point and the diagonally placed motors were hardwired together. Hardwiring ensures identical motor voltage, thus we were able to get close to ideal movement (robot would not rotate as it moves and has nearly straight motion). The robot cannot rotate which is a serious disadvantage. Due to nearly ideal bot movement, only having a wall closeness correction was sufficient.
- Up to Milestone 2:

  A second motor controller was added to allow separate control of each motor and enable the robot to rotate. However, this change made the movement less desirable because the diagonal motors are no longer at identical voltage even when we were giving the same PWM signals to their enable pins. PID control was added to correct for all three types of deviations. A significant amount of time was spent tuning the PID gains, leaving very less time (<4 days) for testing, localization, and block pickup. The robot would move based on commands from matlab via serial communication. The locomotion method was to navigate the maze as in obstacle avoidance and rotate only after reaching one block before target block (gripper towards pickup/drop location). The localization was unreliable because the amount by which it moves in one iteration is unknown. This issue could have been solved with the use of encoders.

Link to a successful instance of probabilistic localization.

- Milestone 2:

  With the unsuccessful implementation of a probability based localization model, the localization was hardcoded. The robot would now move in discrete steps of one block size and always keep the heading towards gripper as required by our algorithm. Also, it would rotate at turns as any other robot (to ensure the heading is always towards the block). Between movements, continuous PID correction for 2 seconds was used. This change slowed the movement of the robot although it significantly improved reliability. The wall follower correction was used to correct the position to a wall after rotation. Communication using Matlab was also avoided by having all our code flashed in Arduino itself, leaving serial communication over bluetooth for debugging.

Comparison with Other Designs

1. The robot size is larger which makes it more prone to hitting walls. Rotating leaves little clearance to the walls.
2. Due to the large size, avoiding obstacles effectively was more difficult and PID correction was implemented.
3. Used object oriented programming, creating libraries for classes and definitions

Future Improvements

1. Lack of time was faced in the final days and could have been avoided.

2. For the purpose of this project, a quick solution to make the robot capable to perform the tasks should have been used, and then optimizing or making a “cleaner”, more general, solution.
3. Should have planned for the robot’s rotation at the time of Milestone 1.
4. Use of encoders for precise movement measuring (could have made the probabilistic approach successful).
5. The successful implement of the probability based localization model with continuous motion.
6. The wiring could be simplified, the ultrasonic echo pins could have been combined as long as the two sensors on each side were controlled separately, only two pins are needed, instead of eight for the ultrasonic echo pins on the sides of the robot. The multiplexer was also redundant after the team switched to Arduino Mega.
7. All of the individual tasks completed and working concurrently for a run through the maze, while not necessary, would have more greatly demonstrated the time put into the robot and the work done by all of the members in a more compact demonstration.

Individual Tasks

Aldrin Dave Perez - Wiring

The goal of clean wiring was to simplify troubleshooting and provide a clean overall look to the rover. The main strategy for achieving clean wiring through the robot was to add wiring accomodations the 3D printed mounts (See Fig. 8) and, wherever possible, run wires through holes in the acrylic base plate to store them underneath the rover. The wires themselves were also all terminated with crimped on 2.54mm Dupont connectors to match the headers on the Arduino and motor controllers. Wires were also kept attached to using multi pin headers instead of running individual wires.

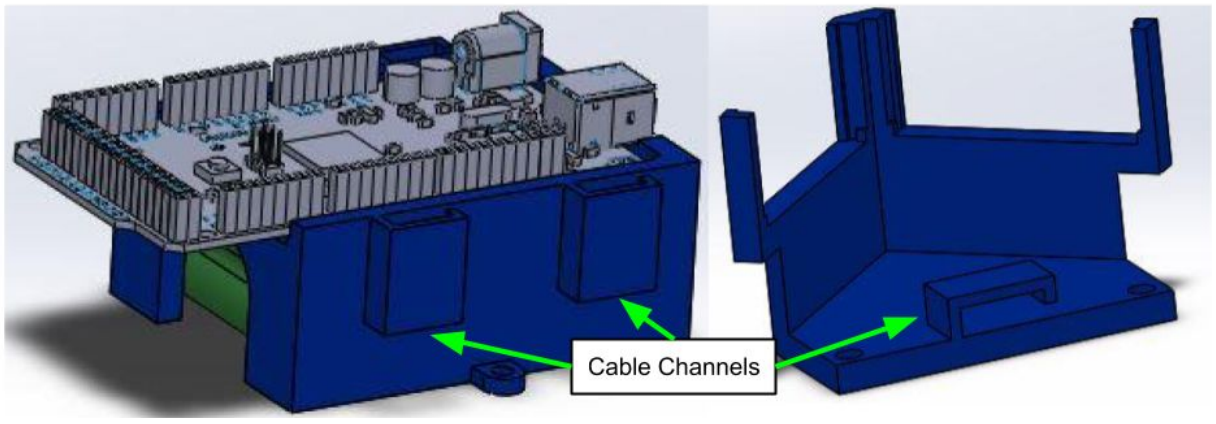


Fig. 8: The Arduino Mega/Battery Holder (left) and Ultrasonic Mount (right) highlighting the cable channels built into them.

The main challenge to wiring the robot was a challenge that the team faced for the overall robot as well. Due to the constant change in strategy and detailed design meant that wiring was changing fairly constantly. New components and wires were added right up to the presentation time, meaning there was no time to clean up the wires. Also, with the use of Arduino Uno and a multiplexer in Milestone 1, the team was unwilling to risk any issues with the switch to no multiplexer, which the Arduino Mega would have allowed.

Unfortunately, the final robot did not manage to achieve the overall clean look due to the above challenges. Many wires were kept above the robot since they had to be added and changed fairly regularly. This did benefit the troubleshooting inadvertently since the wires were easy to access. The use of the multiplexer also meant that there were more wires than necessary, and ultrasonic sensor wires had to run further to the breadboard instead of to the Mega where cable channels were available.

In a future implementation, the main strategy would be to finish the design and testing phase as early as possible to allow for more time to fix the wiring. Also, getting rid of the multiplexer would free up a lot of wiring. The use of coloured heat-shrink tubing would have also helped greatly in identifying the source of a group of wires, as well as keeping the similar wires together for cleaner wire runs.

<u>Jiayu Hu - Find randomly placed block</u>

Block searching sequence:

1. Assume localization is achieved and the rover is at the right entrance of the loading zone with the gripper facing the loading zone.
2. Search the two squares in front of the rover, if block is found subsequent gripper sequence is activated. If not found move close to the wall of the second square and rotate 90 degrees counterclockwise.
3. Rover will conduct a search while moving to its left until it's back to the loading zone entrance.

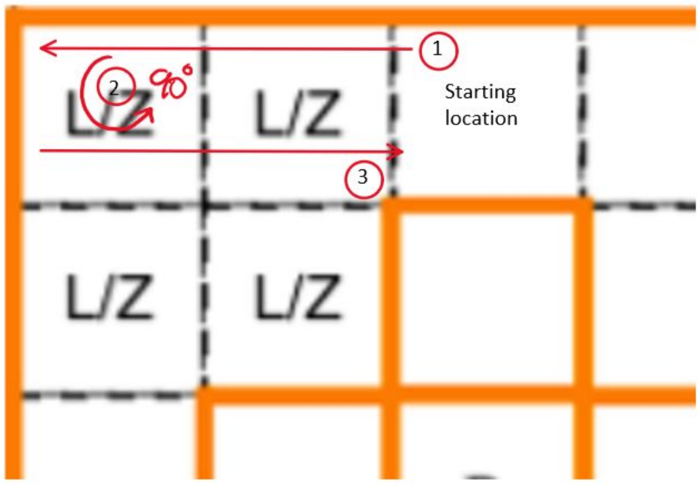


Fig. 9: Block search sequence.

Block search algorithm: originally a servo motor was attached to an ultrasonic sensor to do a sweeping search of the block. However the sweeping ultrasonic sensor had issues detecting the block as the echo signals were deflected by the block if it was at an angle (See Fig. 9).

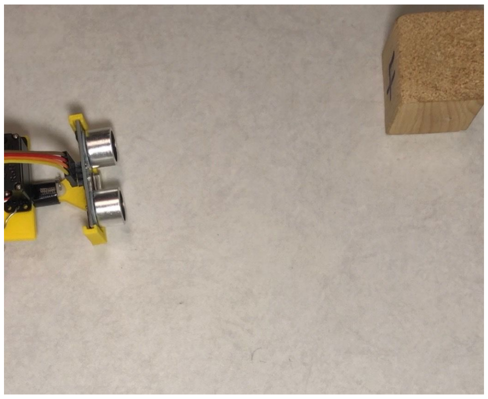


Fig 10: Original ultrasonic sweep search setup.

A more robust search method was implemented with the ultrasonic sensor mounted stationary on the bottom of the rover. The algorithm was program so that it would compare the bottom ultrasonic distance with the top ultrasonic distance. If both sensors output similar distances that means a block is not present and the rover is facing a wall. If the bottom distance is less than the top distance, that means a block is detected and the rover should move forward until it’s close enough for the gripper to grab the block. A demo was shown to Andrew and the rover was able to successfully find and grip the block when placed in 4 random positions inside the loading zone.

Anthony Raso - Sound Effects

The goal of this task was to provide sound effects for actions the robot makes in the maze, as well as effects based on its location or stage in the run. These sounds were implemented by sending single digit numerical values to a computer by printing to serial over Bluetooth (this Bluetooth connection was another members task), where a Matlab program would take the value and determine the sound effect to play which would then be played on the Bluetooth speaker housed on the robot. The two types of effects were achieved at the same time, having the location/stage based effects playing continuously and the action based effects occurring for a short period of time when the action took place. The chosen sound effects can be seen in the two tables below.

Action	Number of Occurrences	Description
Starts Moving	Once	"GO!"
Moves Forward After Changing Direction	Multiple, depends on path	"Forward"
Rotates Left	Multiple, depends on path	"Left"
Rotates Right	Multiple, depends on path	"Right"
Turns Around	Multiple, depends on path	Tape Rewinding Sound
Picks Up Block	Once	"Pokeball Go!"
Drops Off Block	Once	"Complete!"

Location/Stage	Song
Before Localization	*Mad World* by Gary Jules
Localized and on route to block	*Mute City* from the F-Zero Game Franchise
In pick-up zone	Battle Music from Pokemon Diamond
To drop-off zone	*Deja Vu* from the Initial D TV Series
In drop-off zone	*All I Do Is Win* by DJ Khalid, followed by *Sweet Victory* by David Glen Eisley

One of the first challenges was in using Matlab to play more than one audio file at once. In Matlab, I could only find two ways of dealing with an audio file, playing it, and clearing all sounds playing on the speaker and could not pause or do any other manipulation to the files being played. When a file was played, and then another was played afterwords without clearing the song, they would both be playing at the same time on the speaker, and they would continue to compound after every new song was played. The first mitigation to this was to clear the sound when every location/stage based file was played. While this did work, the original sound files, which were longer than five seconds, would play on top of each other or just be cut off when the location/stage changed. As the action effects did occur close to other sound effects, like forward and left or right, I limited them to around one second, fixing the overall compounding and cut-off problem.

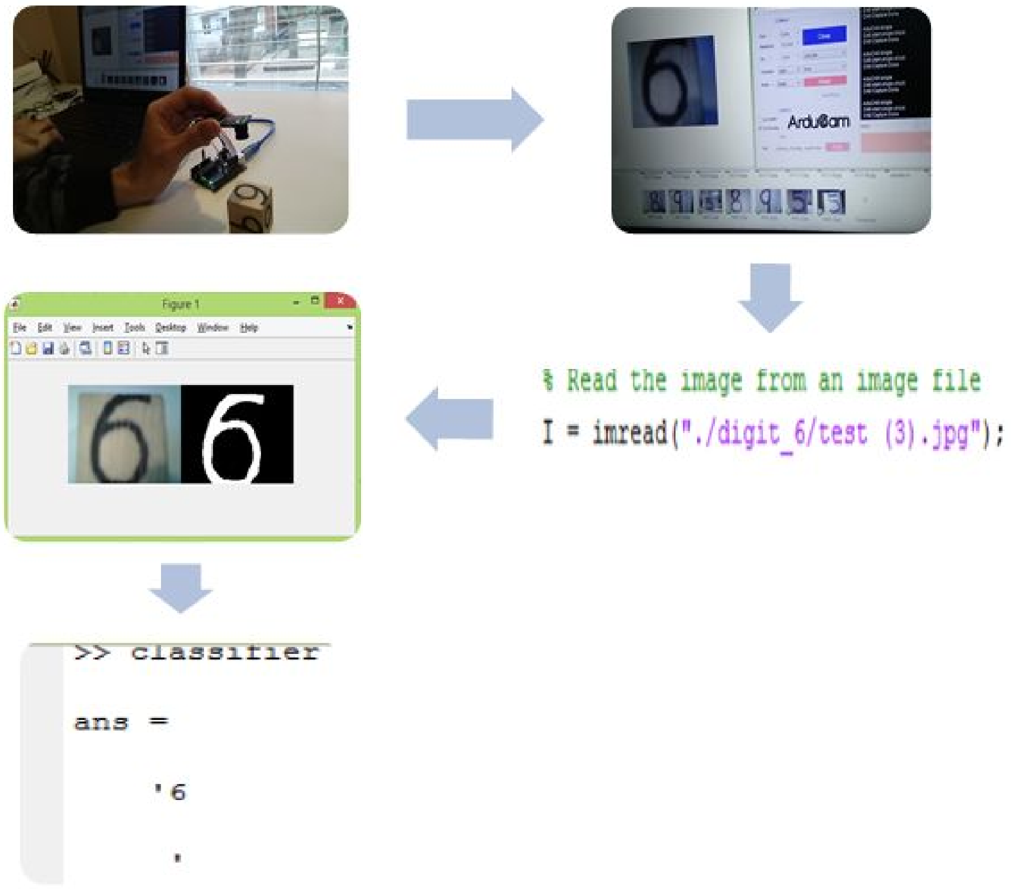
Overall, the sound effects worked when they were supposed to in the limited testing which was available. As the robot was not in working condition until right before the final run, not much time was allowed for testing the effects. There could be more bugs in terms of when and which sounds are played as well as their consistency, which would require setting the robot up in different parts of the maze and doing several runs, which time did not allow. What I might have done differently was to use a buzzer or sound device connected directly to the arduino. I really like the flexibility with the quality of sounds and music i got out the Bluetooth speaker but directly integrating it to arduino would have given more time to debug and make sure everything was in working order to be demonstrated during the final run.

Sumant Bagri - Digit Classification

Augmenting a robot so that it is able to recognize digits/colours on items for pick-and-place processes would be of tremendous help to achieve a specific ordering of items and also eliminate the need of human intervention to do so. The general idea would be to capture the image of the pattern (digit/colour) and do image processing on a relatively smaller scale (so that it is fast and efficient) to give an output that classifies that particular item. Images can be captured using an IR sensor (cheaper but less reliable) or a small camera shield (expensive but highly reliable). When using a camera mounted on the robot a bluetooth connection can be used for wireless communication and data retrieval. In this case image capture is being done manually for demonstration purposes. Most camera modules also provide a “continuous capture” mode (video) which can be used to live stream robot movement. A sequence of classified digits can then be generated and the entire process can thus be organized.

Process Flow:

1) Capture an image using ArduCAM Mini camera module. An open source library for operating this camera is available on GitHub.
2) Captured image is stored in a local folder on the computer.
3) The image is then imported into a Matlab code which does all the preprocessing and generates the classified digit as output.
4) The output can then be further used to generate a sequence of desired ordering to prioritize block pick up in the case of multiple blocks (not our case).

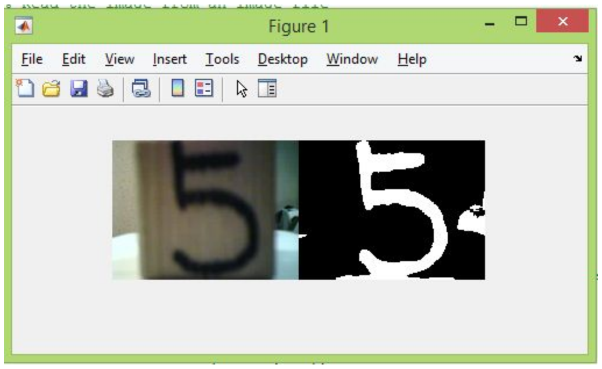


```
% Read the image from an image file
I = imread("./digit_6/test (3).jpg");
```

```
>> classifier

ans =

    '6

     '
```

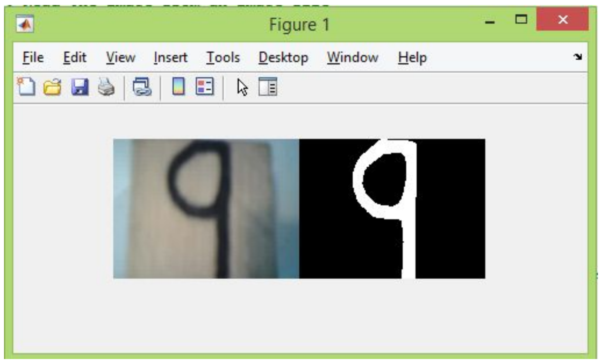
Outputs of other classified digits:

Digit 5:



```
>> classifier

ans =

    '5

     '
```

Digit 9:



```
>> classifier

ans =

    '9

     '
```

Rahul Raj - Upload code through Bluetooth

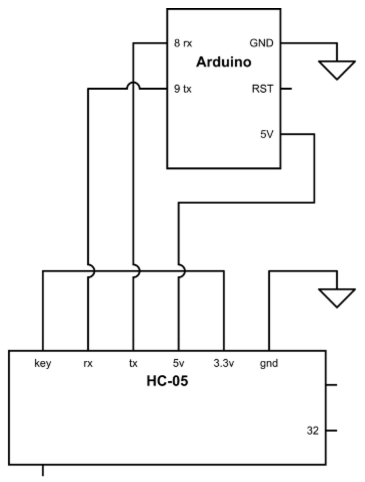
While working with any mobile electronic robot, it becomes really inconvenient to program it or monitor its functionality using a wired connection especially when it has to be done several times. Moreover, it is difficult to keep the usb cable slack throughout the run of the robot so that the attached physical cable doesn't have any influence on the bot's movement. The purpose of the task was to establish a reliable bluetooth connection between the host (computer) and microprocessor of the bot (Arduino). The connection must be capable of flashing new code as support real time serial communication for debugging/monitoring.

Procedure:

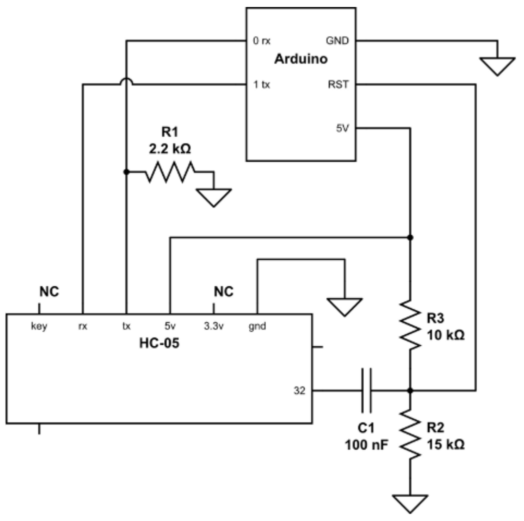
1. To set the bluetooth device name, password and baud rate, we need to program the bluetooth module first. For this, we short, the TX and RX pin of arduino with TX and RX pins of HC-05 respectively. This insures that whatever information we send to arduino is also shared with HC-05.
    a. Power the circuit while pressing key button on HC-05 for a few seconds.
    b. Open a serial monitor on the usb port. HC-05 follows ATtention protocol (common to modems).Enter the following commands, in sequence.

Command	Description
AT	Must return an "OK". Else, check your circuit.
AT+ORGL	Restores the default configuration.
AT+NAME=Name	Setting of desired name of bluetooth device
AT+POLAR=1,0	Setting PIO8 to 1 and PIO9 to 0. (Change led behaviour)
AT+ROLE=0	To restrict the module to only accept connections (slave)
AT+UART=<115200,0,0>	Standard baud rate for communication with UNO/MEGA
AT+PSWD=desired_password	Setting password
AT+INIT	Initialize the module with these parameters.

2. To upload code:
    a. To have a reset of arduino just before beginning code upload, we need rising edge of pulse (low to high). One option (which I followed) is to connect pin 32 of HC-05 with reset pin in series with a capacitor. Due to small size of pin-32, people have figured out other methods using the state pin, but those didn't work out for me and I had to stick to the conventional method.
    b. Just power the circuit, connect your computer to the bluetooth device (with the name you set in step 1). It should open two COM ports on your computer, usually one of them works and other is just an adhoc port. Code uploading should work now using one of the port. Make sure you've set the right board and port in "Tools".



Programming HC-05 (Ref-3)



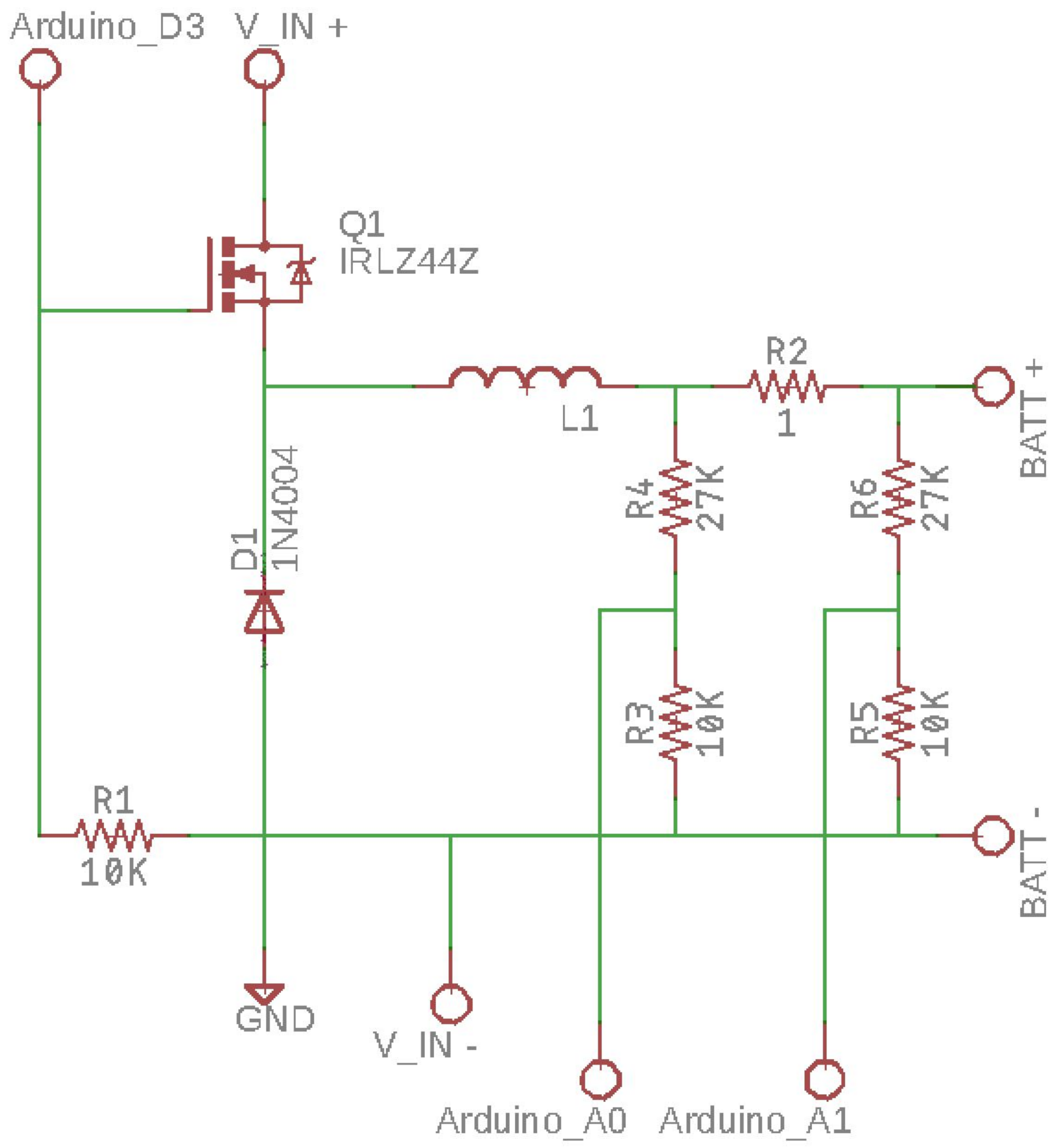
Programming Arduino (Ref-3)

Kyle Mccarroll - NiMH Charging Circuit

Video Link:
https://share.icloud.com/photos/0IdaPXCPsXHSNPAKTCReIobIQ#University_of_Toronto_-_Downtown_Toronto_Campus

A NiMH charging circuit was designed using a MOSFET, as well as basic circuit components (resistors, inductor, jumper wires, etc.), and is controlled by the onboard Arduino. The schematic is shown on the next page. The Arduino in the charging circuit monitors the current using the voltage drop over a 1 ohm 2W resistor and alters a PWM output sent to the MOSFET gate which will alter the charging current. This circuit can charge the battery at 1.4A although the battery should be charged at approximately 1A. The Arduino also monitors the battery voltage to and adjusts the PWM signal to slow and eventually cut the charging current as the battery approaches a full charge. The circuit is loosely based off a buck converter with the capacitor and load replaced by the battery (as the battery will act similar to a capacitor and smooth the voltage). As a result, the charging circuit cannot charge a battery above the supply voltage so the power supply should be set above 15V (voltage of fully charged battery). The MOSFET used in this design can be used for up to 55V supply voltage although the supply voltage should not be set above 20V for safety. The power supply also should have a current limit less than $1.4A \times Battery\ Voltage \div Supply\ Voltage$ set or utilize an in line fuse.

Once the circuit is built according to the schematic and code uploaded to the arduino, connect the grounds for the Arduino, power supply, and battery (common grounds) to the circuit as shown in the schematic. Connect the power supply and battery positive to the points shown in the schematic. Connect the arduino A0, A1, and D3 pins to the points shown in the circuit. Then power on the power supply at a voltage above the maximum battery voltage (15V) with a current limit or in line fuse as specified earlier. The arduino will manage the charging current to prevent overcharging and regulate charging current.



Team Overall Task - Random Starting Orientation

To achieve random starting orientation, the team used the inherent rotation mechanism of the main robot loop. Since the robot already checks which of its four faces is “blocked” (a wall) or “unblocked” (a free, feasible travel direction), achieving a start from a random orientation was as simple as adding a one-time starting check to the first loop of navigation code. This extra check mimics the check in the main loop that looks to see if the current heading is feasible by looking for the average of the pair of ultrasonic sensor readings on the forward face. If the readings are within a certain value, the robot sees that direction as “blocked” and changes the forward heading to another face which has readings that register as unblocked. This check was aided by the random orientations being limited to the four cardinal directions, which keeps all four faces of the robot parallel to the potential walls they would face, limiting any possible corrections in heading to 90 or 180 deg.

This task was achieved with relative ease during the final test run, with the robot being started with the “forward” direction blocked, but quickly realizing its right face was unblocked and moving in that direction instead. The main challenge in realizing this task was that lack of rover rotation control, since it simply used a timer and a fixed rotation speed to always rotate in 90 deg increments. This meant the timer had to be tuned, and was liable to the changing battery voltage. To remedy this, the team made sure to keep the battery voltage within a narrow range to make all motions independent of battery charge levels. The team also implemented a PID correction after every 90 deg rotation increment to keep each face parallel and within a certain distance range to any available walls, ensuring that rover always ended a motion centered and square inside a 12” x 12” block of the maze.

As a future improvement, the team would like to add rotary encoders so the rover can sense how much it has actually rotated, instead of rotating blindly and correcting. The would end the need to keep battery voltages within a tight range and reduce the amount of PID correction the rover would need after any motion, both making it more robust and potential making the code simpler. Adding encoders would also help with the robot’s main 1 block increment motion, since that also uses a similar tuned, “blind” timer and fixed motion speed. The only downsides to this would be small added cost, weight and potentially higher initial setup complexity due to the addition of more sensors that must be wired and calibrated.

Cost Analysis

Below is the final parts list for the robot that completed Milestone 2.

Item	**Qty**	**Purpose**	**Total Price(CAD)**
Arduino Mega 2560	1	Communication and control	21.86[5]
USB Programming Cable	1	Programming communication	1.30[6]
12V NiMH Battery Pack	1	Power source	39.41 [7]
12V DC Motors	4	Drive Omni wheels	67.55[8]
Servo Motor	1	Motor to drive gripper	4.11[9]
L298N Motor Controller	2	Dual-channel controller to drive motors	4.54 [10]
Omni wheels	4	Provide omni-directional movement	41.2[11]
Ultrasonic Sensor	9	Distance sensing, localization	11.08[12]
Multiplexer	1	Read multiple sensor data	4.83 [13]
Bluetooth Adaptor	1	Allow for connection to Arduino via Bluetooth	4.15[14]
Jumper Cables	>50	Allow electrical connection	N/A
3D Printed Parts	>10	Gripper, sensor holders, supports	N/A
Acrylic Sheet	1	Mounting for motors and sensors	N/A
TOTAL			200.03

Contribution List

Robot tasks:

Task	Aldrin	Jiayu	Anthony	Sumant	Rahul	Kyle
Mechanical design	X	X	X			X
Wiring	X					
Locomotion Programming				X	X	
Navigation Programming				X	X	
Localizing				X	X	
Block Pick-up	X	X	X			X
Debugging & Testing	X	X	X	X	X	X

Document sections:

Task	Aldrin	Jiayu	Anthony	Sumant	Rahul	Kyle
Executive Summary			X			
Detailed Rover Design	X					
Maze Solving Strategy				X	X	
Final Results				X		
Discussion		X	X		X	X
Individual Task	X	X	X	X	X	X
Cost Analysis		X				

Final Read Through and Revision	X					X

References

[1]M. &raquo;, "Arduino Programming With Bluetooth", Instructables.com, 2018. [Online]. Available: https://www.instructables.com/id/Arduino-Bluetooth-Programming/. [Accessed: 15- Dec- 2018].

[2]"Control an Arduino with Bluetooth", Allaboutcircuits.com, 2018. [Online]. Available: https://www.allaboutcircuits.com/projects/control-an-arduino-using-your-phone/. [Accessed: 15- Dec- 2018].

[3]J. Newell, "DIY Arduino Bluetooth Programming Shield | Make:", Make: DIY Projects and Ideas for Makers, 2018. [Online]. Available: https://makezine.com/projects/diy-arduino-bluetooth-programming-shield/. [Accessed: 15- Dec- 2018].

[4]"Serial Port Bluetooth Module (Master/Slave) : HC-05 - ITEAD Wiki", Itead.cc, 2018. [Online]. Available: https://www.itead.cc/wiki/Serial_Port_Bluetooth_Module_(Master/Slave)_:_HC-05. [Accessed: 15- Dec- 2018].

[5]Amazon.ca, 2018. [Online]. Available: https://www.amazon.ca/Elegoo-Board-ATmega2560-ATMEGA16U2-Arduino/dp/B01H4ZLZLQ/ref=sr_1_2_sspa?ie=UTF8&qid=1544650995&sr=8-2-spons&keywords=arduino+mega&psc=1. [Accessed: 12- Dec- 2018].

[6]. R3, "Aliexpress.com : Buy 6FT USB 2.0 A B Male Printer Cable 1.8m for Arduino uno R3 from Reliable cable usb 2.0 suppliers on Robotlinking Store", aliexpress.com,

2018. [Online]. Available:
https://www.aliexpress.com/store/product/6FT-USB-2-0-A-B-Male-Printer-Cable-1-8m-for-Arduino-uno-R3/1738188_32473271460.html?ws_ab_test=searchweb0_0,searchweb201602_3_10065_5017220_10068_10130_5017320_10547_10059_10548_10696_100031_10084_5017518_10083_10103_451_10618_452_5017418_10139_10307,searchweb201603_1,ppcSwitch_5_ppcChannel&algo_expid=aeba050e-2790-4597-ac3e-31e8ac07af64-8&algo_pvid=aeba050e-2790-4597-ac3e-31e8ac07af64&transAbTest=ae803_2&priceBeautifyAB=0. [Accessed: 11- Oct- 2018].

[7]"Rechargable 12V NiMH Battery Pack", Robotshop, 2018. [Online]. Available:
https://www.robotshop.com/ca/en/12v-1600mah-rechargeable-nimh-battery.html.
[Accessed: 11- Oct- 2018].

[8]"Robotshop", 12V 100RPM 583 oz-in Brushed DC Motor, 2018. [Online]. Available:
https://www.robotshop.com/en/12v-100rpm-583-oz-in-brushed-dc-motor.html.
[Accessed: 11- Oct- 2018].

[9]"1pcs/lot MG995 55g servos Digital Metal Gear rc car robot Servo-in Integrated Circuits from Electronic Components & Supplies on Aliexpress.com | Alibaba Group", aliexpress.com, 2018. [Online]. Available:
https://www.aliexpress.com/item/Free-shipping-1pcs-lot-MG995-55g-servos-Digital-Metal-Gear-rc-car-robot-Servo/32355876998.html. [Accessed: 11- Oct- 2018].

[10]"L298N L298 25W Dual Channel H Bridge DC Stepper Motor L298N Drive Controller Board Module 5V 2A Driver Control For Arduino-in Motor Controller from Home Improvement on Aliexpress.com | Alibaba Group", aliexpress.com, 2018. [Online]. Available:
https://www.aliexpress.com/item/L298N-L298-25W-Dual-Channel-H-Bridge-DC-Stepper-Motor-L298N-Drive-Controller-Board-Module-5V/32842834591.html?spm=2114.search0104.3.1.2038476bmf9JuJ&ws_ab_test=searchweb0_0,searchweb201602_4_10065_10130_10068_318_10547_319_5727317_10548_10696_10084_10083_10618_452_10139_10307_532_204_10059_10884_10887_100031_320_10103_5727217,searchweb201603_1,ppcSwitch_0&algo_expid=aff9b669-5157-4917-b1b0-6a2b1972bee8-0&algo_pvid=aff9b669-5157-4917-b1b0-6a2b1972bee8&priceBeautifyAB=0. [Accessed: 11- Oct- 2018].

[11]"58mm Plastic Omni Wheel (compatible with Servos and Lego Mindstorms NXT)", 2018. [Online]. Available:
https://www.robotshop.com/ca/en/58mm-plastic-omni-wheel-compatible-servos-lego-mi

ndstorms-nxt.html. [Accessed: 11- Oct- 2018].

[12] "5pcs/lot HC SR04 ultrasonic ranging module smart car ultrasonic sensor module-in Sensors from Electronic Components & Supplies on Aliexpress.com | Alibaba Group", aliexpress.com, 2018. [Online]. Available: https://www.aliexpress.com/item/5pcs-lot-HC-SR04-ultrasonic-ranging-module-smart-car-ultrasonic-sensor-module/32531458441.html. [Accessed: 12- Oct- 2018].

[13]"16 Channel 1 x 16:1 Multiplexer Switch ICs | Mouser Canada", Mouser.ca, 2018. [Online]. Available: https://www.mouser.ca/Semiconductors/Integrated-Circuits-ICs/Switch-ICs/Multiplexer-Switch-ICs/_/N-7591t?P=1yp2k8sZ1yzvta4. [Accessed: 12- Dec- 2018].

[14]"Bluetooth Adapter V4.0 CSR Dual Mode Wireless Mini USB Bluetooth Dongle 4.0 Transmitter for Computer PC-in USB Bluetooth Adapters/Dongles from Computer & Office on Aliexpress.com | Alibaba Group", aliexpress.com, 2018. [Online]. Available: https://www.aliexpress.com/item/Bluetooth-Adapter-V4-0-CSR-Dual-Mode-Wireless-Mini-USB-Bluetooth-Dongle-4-0-Transmitter-For/32817286984.html. [Accessed: 12- Oct- 2018].

Appendix

Localization using Probability maps

The world is discretized with a block size of 12’’X12’’ spaced by 6’’, which means that two adjacent blocks are overlapping by 6 inches. If we count the blocks it is 15 blocks (8 +7 overlapping) horizontally and 7 (4 +3 overlapping) vertically. Each of these positions correspond to 4 states, one for each orientation. Each of these states store an 8 length array.

When the robot is placed in the world it maps the states as obtained from the sensor readings to the states as described above and assigns a probability value to where the bot could or could not be. As the bot moves in the maze this map keeps updating with the probability values and robot belief of its actual position becomes stronger (highest probability value). Once the “belief” becomes strong enough (higher than a threshold probability value) the robot is assumed to be localized.