Rose-Hulman Institute of Technology

Mobile Robotics: ECE425

Navigation Competencies

Date: 2/19/2022

To: Professor Berry

Team: LEONARDO

# Abstract

This project combined the all previous skills this quarter to have LEONARDO maneuver a 4x4 world. LEO is equipped with four IR sensors, two sonar sensors, a GY-521 IMU, and used a differential drive with two active wheels and a caster to balance it. The controller for LEO was an Arduino Mega used to drive the motor via the AccelStepper library and a HC-05 Bluetooth module was implemented for additional communication and computing power to be done remotely. The major four tasks in this project were: topological path planning, metric map path planning, localization, and mapping. The robot maneuvered the 4x4 world using an occupancy grid and a topological map for each of the tasks.

Table of Contents

[Abstract 2](#_Toc96266773)

[I. Objectives 4](#_Toc96266774)

[II. Theory 4](#_Toc96266775)

[III. Methods 5](#_Toc96266776)

[IV. Results 7](#_Toc96266777)

[V. Conclusions and Recommendations 7](#_Toc96266778)

[References 8](#_Toc96266779)

[Appendix/Supplementary Materials 9](#_Toc96266780)

# I. Objectives

This project is to have LEO path plan, localize and map the given 4x4 world based upon both an occupancy grid and topological map. LEO maneuvered the world from a set start and end point using topological or metric pathing commands. The localization task used topological features of the map for LEO to detect his current position within the map. The mapping task had LEO know his set point and from that position map his environment based upon sensed information as it transverses the world. There is also the additional task of Simulations Localization And Mapping, but this task is optional.

# II. Theory

The theory used for path planning mostly based upon a recursive A\* algorithm. In the practical case of path planning a world, we used a wavefront propagation-based form A\* since A\* is most commonly used with nodes rather than coordinates. Starting from the end point, each block surrounding that block would be given a number that is larger than the previous number. This pathing would then propagate through the map and stop until the start point is reached. The pathing algorithm accounted for topological obstacles that would inhibit LEO from traversing through the map. In theory, this should map out all shortest paths LEO could take to reach the end point. Both topological and metric path planning were based upon the A\* pathed map, but different commands were used to differentiate topological and metric pathing.

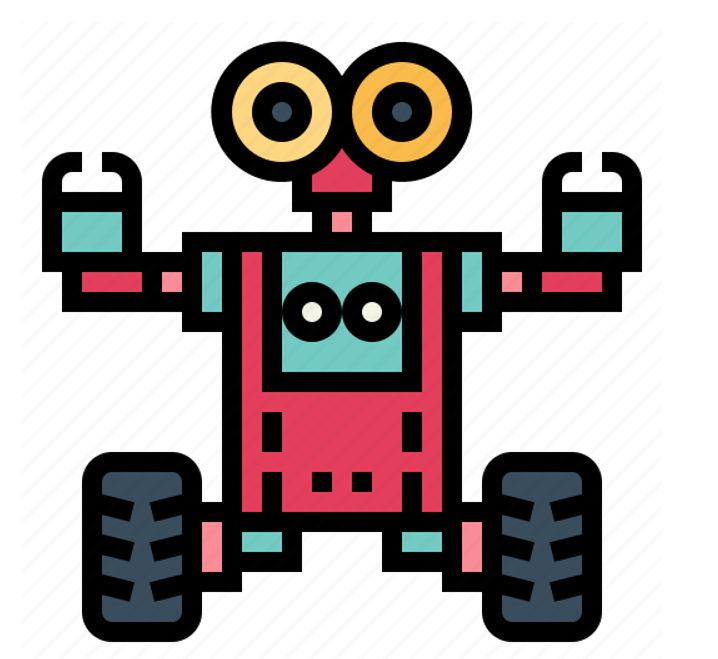
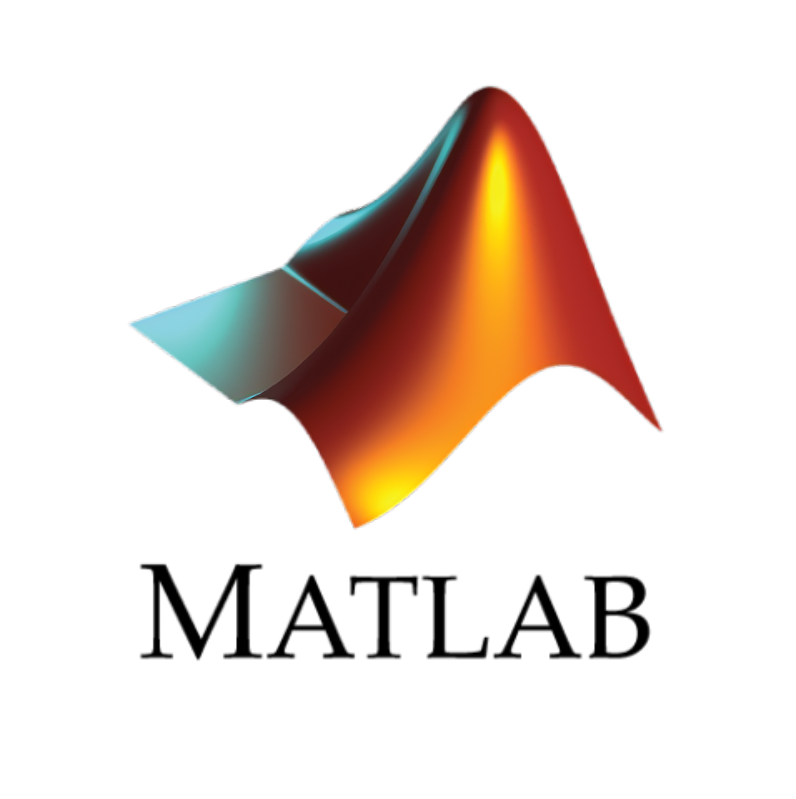
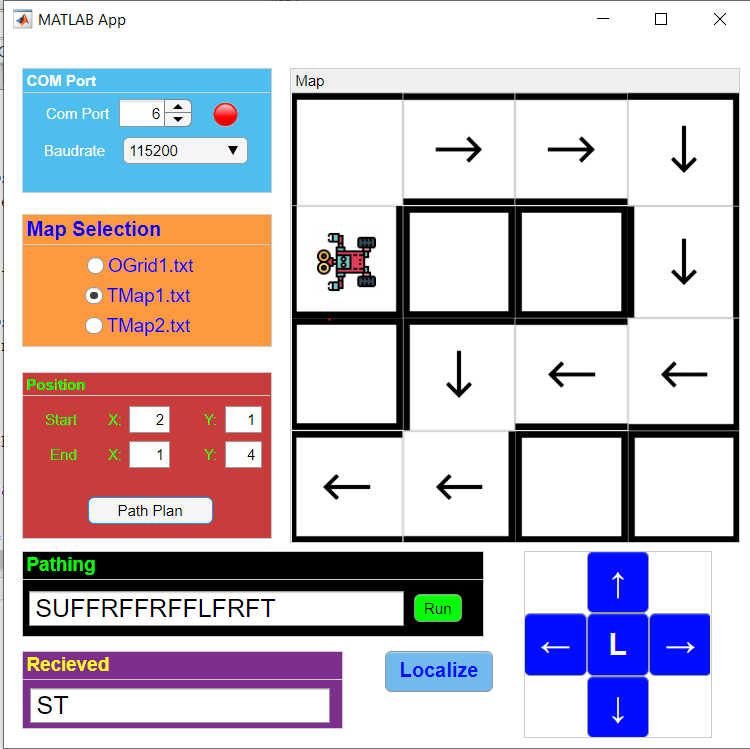
The theory used for localization was to use a given map and have LEO ping the walls with the IR sensors to find the corresponding block that correlates with the ping. If there ends up being multiple blocks that correlates with the ping, LEO would relocate to an adjacent opening to a new block. LEO would then remember the previous block value and his previous pathing. From that, every new ping would then transverse LEO’s old path and fit that path of blocks onto the given map.

The theory used for mapping was to have a preset start point and orientation and have LEO ping each block he transverses through. The traversing algorithm is to follow the left wall of every block to transverse most of the map. Some parts of the map LEO would need to revisit to increase confidence on block numbers. The initial map LEO would assume is that every block is a blocked path until proven otherwise using the IR sensors to ping each wall.

The theory for SLAM, an additional optional objective, would be to have LEO use the mapping algorithm to drive most of the initial motion. LEO would also use the localization algorithm to set coordinates to the mapped location from its start point. Once the world has been mapped and localized LEO would return to his start position and execute finding the light beacon in the map using a modified localization-based algorithm. LEO would then use a modified mapping-based algorithm to compare the previous map to the current map to find any changes to the map for the exit.

# III. Methods

A high-level summary of the robot architecture was to have the GUI be used for only human interfacing and task commands sent to the Matlab object, a Matlab object be used to do intense computation (A\*, localization and mapping) and send commands to LEO, LEO receives and executes command via Bluetooth and returning a ping to Matlab when finishing the command. This form of architectures reduces the amount of computation in the Arduino and uses Matlab instead. It also allows for almost conversationally commanding the LEO because the GUI would update almost real time after each command is passed. It also allows for easy troubleshooting and identifying which aspect from this architecture has bugs and errors. Further documentation of implementing the Bluetooth module can be found in Appendix A.



Button Commands

Gcode Commands

Finish Command

Updated Map

*Figure 1: A high-level diagram of the robot architecture working with the GUI, MatLab and LEO robot communicates and executes.*

In order for each element to interact, mainly Matlab and Arduino, the Bluetooth module is utilized with LEO programmed to interpret a simplified flavor of a Gcode. The Gcode interpreter on LEO only understand the following commands: L (Left turn), R (Right turn), F (Forward), U (U-turn), C (Check walls). This interpreter works best for metric path planning, localization and mapping. For topological path planning, the interpreter used a string input of the SLRT format and had LEO process the string as traversing through the world. All tasks are tested for a topological map then tested for an occupancy grid map.

For the path planning task, the high-level overview is to have the GUI interpret the map and send the map into the MatLab object. The Matlab object then creates a map in the form of the wavefront propagation. The object then transverses through the map as if it was LEO and takes the largest gradient descent from the start point to end point. While traversing, a string of commands is being built of forwards and turns to then send commands to LEO. The string is then sent back to the GUI to then update all pathing information. The GUI has an execute command that will take the string and feed it through the object by string for topological or character for metric path planning. The object will then send the command to LEO and await a response. LEO would execute the command and return a response to the object. The object will then resume execution and return to the GUI. The GUI would finally update the map and any other fields to display the robot traversing through the map. The topological commands used the absolution coordinate system in reference to the map. The metric character commands are based upon the topological commands (SLRT) but are intended to be sent character by character using relative coordinates to the LEO’s position and orientation, hence the addition to the forward and U-turn commands.

For the A\* algorithm, implemented in the MatLab object, the end point was set to 0 and propagate from there. A recursive function would then traverse through each point adding one additional value to the block. The information propagated through the function would be the A\* map, the current point it would be traversing, the current value of the block, and a map of visited blocks. This implementation is not optimized for speed, so it most likely traverses through the entire map 256 times due to the recursive nature, it is never counted but would transverse the map multiple times. Through each recursion, the value of each block is checked to be the lowest possible value it could be and check for surrounding blocks to recurse over according to the topological map and the map of visited block. The resulting A\* map gives the shortest possible path following the largest gradient descent. This algorithm follows every possible path LEO could take to reach the end point and the resulting A\* map would include the shortest path.

The path planning algorithm differed between topological and metric, but also used the same algorithm. Topological traverses the A\* map from LEO’s start point and finds a lower value than the current block and adds L or R to a command string when making turns. S and T are tagged on to the command string’s start and end respectively. Metric traverses the A\* map in the same method as topological, but adds L, R, U or F to the command string. L, R, and U are for reorientation of LEO’s reference frame and F is for translation of LEO’s reference frame. Both methods use a lower value block to traverse through the A\* map, and accounts for blocked paths. The issue is that it does not use the largest gradient descent to traverse and result in nearly the shortest path. This is more apparent in open maps with the freedom to wander. LEO’s pathing will almost go in a circle before traversing to the end point.

Localization utilizes the same high-level connectivity of metric path planning, but the MatLab object uses the localization algorithm. Localization would ping LEO for wall updates and the MatLab object would update the current possible localized position. The pings would repeat to LEO from the MatLab object until a single position can be found. When multiple positions can be localized, LEO would then move to another open position through the object sending commands. The object would remember the path and block number. Through each consecutive move command, prior to knowing the localized position, the object would transverse through the map that fits the previous movements. This is similar to fitting a pathed out puzzled piece into the puzzle of the map.

Mapping has not been implemented in code, but the methodology and theory has been worked out. The mapping algorithm works similarly to localization with how commands are being sent from LEO to the MatLab object, but also sending the information from the object to the GUI to update walls. LEO would ideally run automously while mapping the terrain, but if the transverse algorithm proved to be too difficult, LEO would be teleoperated for his transverse algorithm.

# IV. Results

The path planning algorithms and executions were successful. The GUI was able to send the MatLab object interfacing commands with the user. The object the successfully created a path to the end point and return the command string. The command string was then sent or parsed then sent to LEO to then execute. LEO proved to traverse the map and follow the commands properly to result in the end position. The GUI was able to update based off LEO’s and the MatLab object’s execution. For path planning, everything worked as intended and was able to complete the objective as intended. LEO traversed the map from a start point to an end point in both an occupancy grid map and a topological map. The only issue was that the GUI was not fully developed during demonstration and hard to follow the path planning algorithm in action. Furthermore, references to the GUI can be found in Appendix B.

Currently, localization is being worked on and having issues with using the recorded path to localize. Single, unique blocks are able to be localized because there is no recorded path to outline in the map. The GUI is able to send interfacing commands to the MatLab object and the MatLab object can send commands to LEO. The object’s algorithm seems to be failing and have LEO move, but cannot localize based off his moves. This most likely be due to reference frame transformation between LEO’s reference frame and the map’s absolute reference frame.

The mapping algorithm is only a theory and currently being implemented. There are no results from this algorithm as of the time this report is turned in.

# V. Conclusions and Recommendations

Overall, LEO was able to successfully execute the path planning section of the project. The GUI and MatLab interfaces well with LEO in executing the tasks. The high-level planning of the GUI for human interface, MatLab algorithm for solving and LEO interpreter for robot execution proved to be highly versatile for many of the tasks with easy to add algorithms and troubleshooting. Localization is close to being successful, but certain issues with the localization algorithms proved to be problematic. This is most likely due to robot orientation in the absolute orientation. Mapping is currently a theory but utilized previous functions for implementation. The only major change is to add the theory and algorithm in the MatLab object. Future improvements are mostly in optimizing algorithms. The path planning algorithm uses an A\* map but has difficulty detecting the largest gradient descent. Localization algorithm would be optimized in fitting the puzzle piece of the previous visited placed onto the map blocks. This would make localization successful on a much larger scale. The mapping algorithm still needs to be implemented for further improvements. Asides from algorithm, LEO does not have wall-following or obstacle avoidance implemented and would prove to be useful to counter act odometry. The IMU was also not implemented but would prove to be useful for odometery. Overall, LEO has successful completed the path planning task and currently debugging the localization task and implementing the mapping task.

# Questions

1. Were there any issues with the wireless communication? How could you resolve them?  
   If at all.

We experienced some issue related to Bluetooth module. The major problems are related to AT mode and connection. The detailed information is in appendix A.

1. What does the state machine, subsumption architecture, flowchart, or pseudocode look  
   like for the path planning, localization, and mapping? (It should be in the appendix of  
   the report).

Appendix C

1. How would you implement SLAM on the CEENBot given what you have learned about  
   navigation competencies after completing the final project? If you research solutions,  
   make sure you cite and list references in APA or MLA format.
2. What was the strategy for implementing the wavefront algorithm?
3. Were there any points during the navigation when the robot got stuck? If so, how did  
   you extract the robot from that situation?

The robot might collide to the wall during navigation process because we cannot implement the wall following function to it and the odometer error will interfere with the movement of robot. The solution is manually correct the robot movement.

1. How long did it take for the robot to move from the start position to the goal?

It depends on the distance between the two points. For the conner-to-conner situation, it needs about 70 seconds to finish the planning process and movement.

1. What type of algorithm did you use to selection the most optimal or efficient path?
2. How did you represent the robot’s start and goal position at run time?
3. Do you have any recommendations for improving that robot’s navigation or wavefront  
   algorithm?
4. How did you use the serial monitor and bi-directional wireless communication to  
   represent the map?

Most of the calculation is done in the computer, which for the path planning and localization, only the current state of robot will be sent to the computer. The computer will recognize the relationship between map and robot and provide commands to the robot. For mapping, the robot will process the map information and transfer the updated map to the computer through USB cable or Bluetooth connection.

1. What type of map did you create and why?

For path planning and localization, both topological and metric map can be recognized by the Matlab function. For the mapping, the metric map will be used to generate the map because it can show the single wall on the map.

1. What was key in the integration of the localization, mapping, and path planning?

The main key is the good communication and timing between the computer and the robot. Because some tasks need both computers to work together, the good communication and timing became important because bad communication and timing could cause the one computer cannot follow on another one and create errors in processing.

# References

1. Pokorny, Kian & Vincent, Ryan. (2013). Multiple constraint satisfaction problems using the A-star (A\*) search algorithm: classroom scheduling with preferences. Journal of Computing Sciences in Colleges. 28. 152-159.
2. Ratul, MD & Azimi, Saiful & Zainal Abidin, Mohamad Shukri & Ayop, Razman. (2021). Design and Development of GMapping based SLAM Algorithm in Virtual Agricultural Environment. 109-113. 10.1109/ICCSCE52189.2021.9530991.

# Appendix/Supplementary Materials

[**Appendix A – Bluetooth Module** 10](#_Toc96266781)

[x**Appendix B - MatLab GUI** 11](#_Toc96266782)

**Appendix A – Bluetooth Module**

The HC-05 Bluetooth module proved to be difficult to implement.

The first issue encountered was setting the module in AT mode. The module did not have a KEY pin, so many references proved to be difficult to use. The solution was to hold the button down while powering the module. Once powered the light will slowly blink (about 2 second intervals) compared the rapid normal blinking.

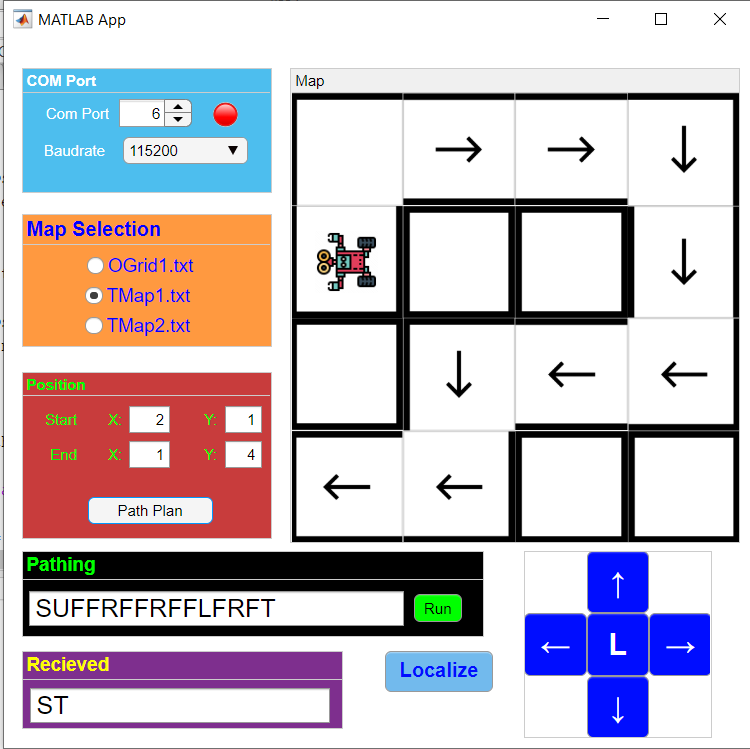
The second issue is how the implement the AT commands. There are provided sketches to upload to the Arduino and connect to the Arduino’s com port communication. The serial monitor communication will then be set the baudrate in the sketch and the settings are changed from “new line” to “NL & CR”. The serial monitor then can communicate to the Bluetooth module using AT commands. The name and Baud rates were changed.

The third issue is that the module had quite a process for connecting. The connection would be done over computer. The module is turned on with the rapid blinking red light (indicating it is looking for device to connect to). The computer will then need to search for the Bluetooth device and connect to the new named module (“LEO” for LEO). After that (LED should be slowly blinking now), serial communication is checked via running the GUI or serial monitor through Arduino IDE. The most frustrating part is that the com port for the Bluetooth module has 2 ports (receiving and a send/receiving). The higher number com port proved to be the successful one and the other port usually freezes the IDE. Once the com port is connected (the LED should be slow blinking), it would need to be disconnected (the LED would turn to a rapid blinking again) for another application to use it (MatLab vs IDE). Also, once the robot is power cycled and the module is reset, the device then has to be forgotten on the computer and reconnected again to ensure there is proper connection (repeat connections and verified through the LED blinking).

x**Appendix B - MatLab GUI**

Map Panel:

Displays current map with obstacles and robot. Has arrows to show planned path



Received Panel:

Currently does nothing. Return received information from LEO from the check walls command

Teleoperation Panel:

Currently does nothing. Allows user to drive LEO for mapping.

Map Selection Panel:

Map selected based off text file name

Localize Button:

Run localization algorithm

Pathing Panel:

Command created from path planning algorithm ready to send to LEO

Position Panel:

Set start and end point for path planning

Com Port Panel:

Port # and Baudrate set