Hayne Kim (hikim2)
Devin Tark (devinvt2)
Nidia Bucarelli (nidiaib2)

**MP3 Project Report**

In this MP, we are using 4 threads as follows.

1. The first thread will be dealing with the safety of the robot, where we are giving the highest priority (4). This thread will stop the robot immediately if any of the safety-related sensors (bumpers/cliff) have detected an event.
2. The second thread will be taking care of the motion control, tracking, navigating, and contouring after the robot has finished the maze. It will help the robot to follow the wall and track the path/trajectory it has taken that will be later drawn out in the contouring part. This thread has been given the second-highest priority (3).
3. The third thread will be performing the computer vision, where the robot will take the picture and save all the pictures in the vector to match the frames against the provided pictures to identify the objects after the robot finishes traversing the maze. This thread will be given the lowest priority (2).
4. The fourth thread will be processing the images taken during navigation. First, "RunObjectIdentification" function is run to determine if the image taken is the lamp. If the detected image is a lamp, the red lights are triggered only once. Once the lamp is detected, the lamp image is deleted when running object identification for the next images. This thread will be given the second- highest priority (3).

The architecture of this MP is summarized in Figure 1.

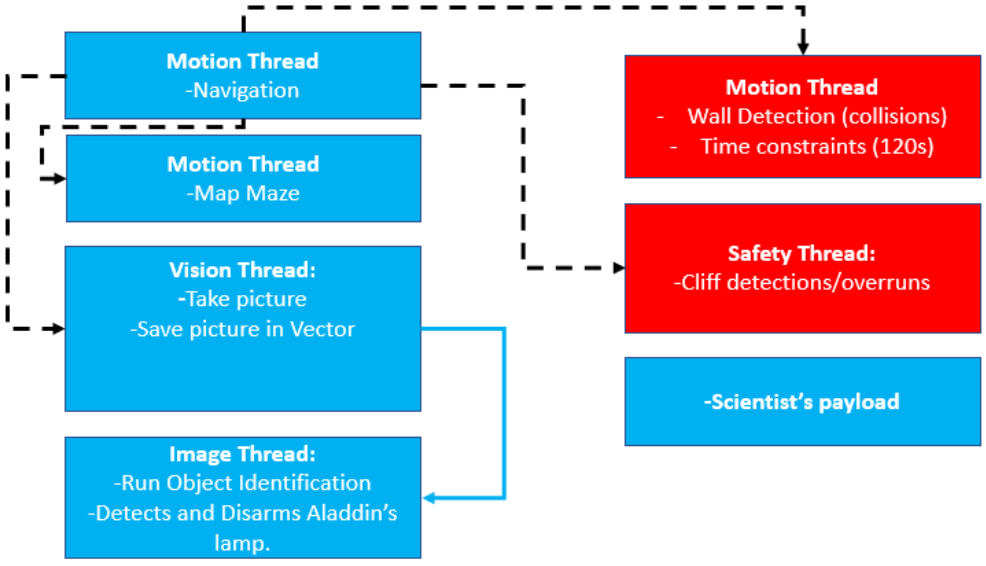



Figure 1

This MP was designed to meet the “mission-critical”, “safety” and “performance” requirements as requested. The Table 1 summarizes the components, functions and critical levels of these requirements.

Table 1

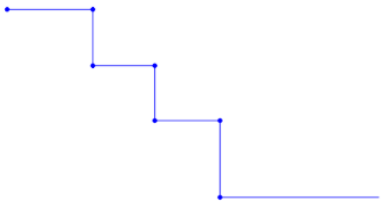
| **Requirements** | **Component Name and Function** | **Critical Level** |
|---|---|---|
| Wall collisions | Motion Thread: the function is invoked when the wall signals detect something. The robot stops, and then turn. | Motion thread priority: 3 |
| Cliffs | Safety Thread: the function is invoked when the cliff right/left signals are higher than 100. The robot stops and play a song. | Safety thread priority: 4 |
| Timing constraints | Motion Thread: a while loop is used to manage the timing constraint. The robot will run only while progTime < 120000. After that, the robot will stop. | Motion thread priority: 3 |
| Map the maze | Motion Thread. When navigating, the distances and angles are saved. Once the navigation finishes, the information is sent to the contour function. | Motion thread priority: 3 |
| Identify encountered objects | Image Thread:<br>The function is invoked when there is an image in the vector from the vision thread. The “Run Object Identification” function identifies the image. | Image thread priority: 3 |
| Disarm Aladdin’s Lamp | Image Thread:<br>The function is invoked when there is an image in the vector from the vision thread. The “Run Object Identification” determines if the image taken is the “Aladdin’s Lamp”. If it is the lamp, red light turns “ON” for 2 seconds. | Image thread priority: 3 |
| Run scientist’s payload | No thread. It’s a component itself. | |
| Time to traverse maze | n/a | |
| Time to finish entire of the maze | n/a | |

**Problems:**
Some major issues that we have encountered were synchronization and architecture/design. We carefully designed our architecture and considered all the race conditions for the three threads and added all the mutex locks where we think we need to. Another issue was optimizing the vision thread my determining when to take images at the optimal time. We decided the optimal time was while "wall turning" because the camera should get 2 good views of the interior maze during this maneuver. We were able to get down the number of images taken down to ~16.

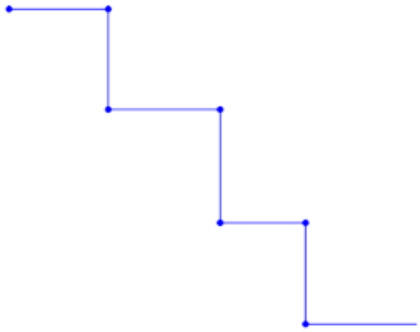
By default, our robot has a speed of 200 mm/s, and it follows all the walls.

**Algorithm for Navigation:**
The motion thread continuously monitors for bumps, as well as if the robot's wall sensor is reading values outside the predetermined corridor of wall sensor values (3 - 90). If the robot reads 6 wall sensor values that average to less than 1, it assumes it has lost a wall and turns approx 90 degrees, then drives forward, and then turns another 90 degrees until it finds another wall. If it reads a sensor outside of the corridor, it turns slightly either towards or away from the wall. If it bumps, it turns counterclockwise, determines the local maximum wall sensor value in order to determine its parallel direction, and then drives straight.

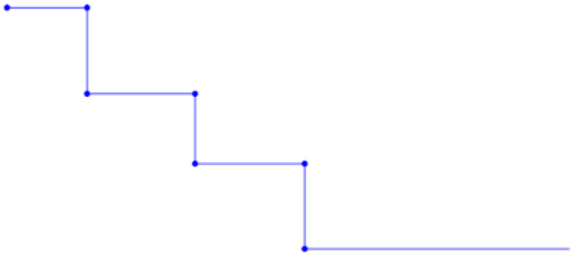
**Examples of Contours:**
We only tested the robot at the lab, and here are the three different examples of contour plot:



**Time**: *65 seconds*



**Time**: *60 seconds*



**Time**: *60 seconds*