MP3: Tomb Raider Robot

Bryant Zhao, Andrew Zhang, Aniket Shirke, Eric Zheng

Design Paradigm

In this MP, we restructured our code for MP2 and made a new architecture to ensure thread-safety and functionality. We reduced the clock period from MP2 to ensure **safety-critical** requirements, divided each modular task into separate threads, rewrote the entire navigation handler as our **mission-critical** module, and fixed object-identification to meet the **performance-critical** requirements.

With each of the 3 primary requirements each being refactored into independent threads, we can now easily maintain well-formed dependencies. We used rate-monotonic scheduling policy to determine thread priorities based on the lengths of their periods. The priority ordering of threads is as follows (with higher values denote higher thread priority):

1) Safety thread – Priority 5, T = 15ms - **safety-critical module**
2) Navigation thread – Priority 3, T = 15ms - **mission-critical module**
3) Object Detection thread: Priority 1, T = 3600ms - **performance-critical module**

The contour mapping module is embedded in the main thread. With a global queue collecting all motion state changes, the contour-mapper records all necessary waypoints during the mission and draws the map at the end of our main thread. Detailed designs will be discussed in the sections below.

Dependency Graph

The dependency graph remains the same as from MP2. No major scheduling or priority changes were made to accomodate MP3.

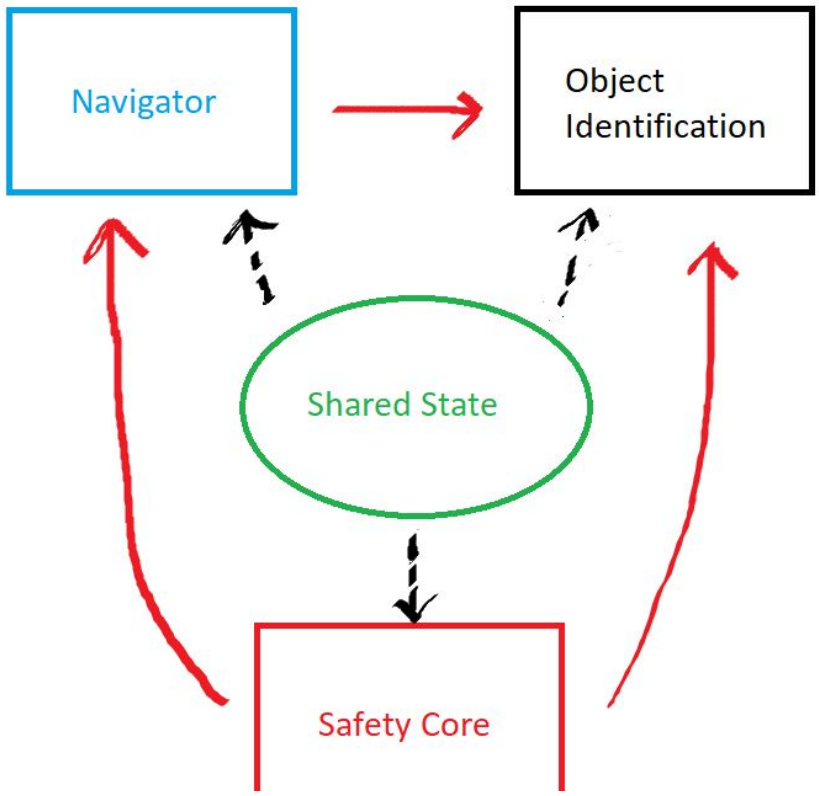


Table of Components

Requirement	Component Name & Functionality	Criticality Level
Disarm Aladdin's Lamp	Object Detection: invoked when the Lamp is detected and it turns the red LED on	Mission-Critical
Identify encountered objects:	Object Detection: take pictures during traversal and process images at the end of the mission	Mission-Critical
Map the maze	Navigator: provide optimized course for the robot throughout the mission	Mission-Critical
Run scientists' payload	<External>	Mission-Critical
Avoid Hard Wall Collisions	Navigator: reduce speed if the robot is about to bump	Safety-Critical
Avoid Falling off Cliffs	Safety Core: triggered if wheel falls detected	Safety-Critical
Avoid exceeding Time Constraints	Safety Core: reprioritize tasks or abort if the length of the mission is about to reach 120s	Safety-Critical
Safe Mission Abort	Safety Core: triggered if the mission goes above 120s	Safety-Critical
Time to traverse maze	Navigator: ensure an optimal speed for the robot to traverse through maze quickly	Performance-Critical
Time to finish entire mission	Navigator, Object Identification, and Contour Mapper: object identification and contour mapping are done at the end of traversal	Performance-Critical

Safety-critical Module

1) *Avoid hard wall collision, avoid falling off cliffs*

   The safety-critical module did not change with respect to the original MP2 safety concerns & response methods.

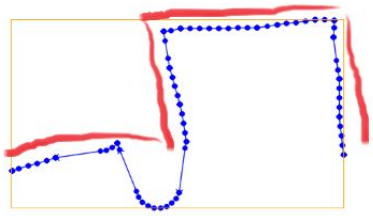
2) *Meeting the 120s timing constraint*

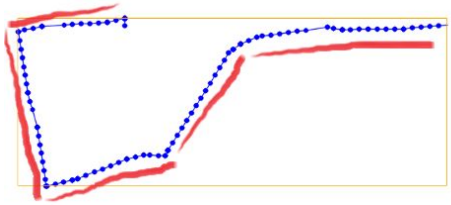
The safety-critical module timed out the 120s hard timing such that if it did occur, it would suspend the robot on it's own, forcing it into a DONE state and performing e.g. contour mapping and image detection. This also enabled the abort option, if desired.

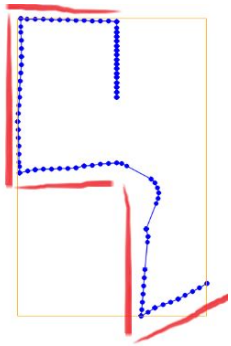
Mission-critical Module

1) *Contour-mapping*

The contour mapping remained the same as from MP2. Below are 3 of the contours taken earlier on via MP2 code, using the slabs from the lab room, ~25-30s per, zoomed in on certain portions for detail:







2) *Image Detection*

The life of a picture taken by the robot is as follows:

1) Once the image is taken, the image is passed through a heuristic function to decide whether we want to keep the image
2) If we decide to keep the image, we segment the image in real time into large regions of contiguous white pixels and store those segments of the images as separate files
3) Upon termination of the program we load all the images and run detection based on SURF descriptors and homographies.

Our heuristic function was formulated because we realized that, by setup, the objects are printed on white paper taped on a white board. Thus, we figured that a picture may contain an object if and only if the picture has a significant portion of white pixels. We introduce a heuristic which counts the number of white pixels and compared it to some empirically tuned threshold. We found it more necessary to underfilter than overfilter.

We decided to segment the image because we realized that in many cases, we didn't need a lot of the information of the image but we wanted to maintain the resolution of the image to maintain as much features as possible. For example, if there were pictures which contained a picture only in 1/8th of the photo, ⅞ of the

photo is pretty much useless. Thus we attempted to find large bounding boxes over contiguous portions of white pixels, and cropped to those bounding rectangles.

3) *Lamp Defusal*

The descriptors and keypoints for 'magic-lamp-600.jpg' are maintained and as soon as an image is taken, we match the descriptors and keypoints of the scene image with the lamp's. Upon detection, we light the red LED for two seconds and then stop the defusing process.

4) *External Tasks*

The external tasks were tested and given adequate time in order to complete ~25 cycles before exiting the maze. This was tested in conjunction with navigation, and some of the heavier concurrent image detection (the lamp). There were some difficulties in working around the existing performance and tuning, and those are discussed further in 'Issues Encountered'.

Performance-critical Module

1) *Timing on Maze & Mission*

The core navigation algorithm remained the same as from MP2 (max speed = 175mm/s), barring some small tuning tweaks aimed more at original MP2 flaws than MP3 needs. However, there was 1 major modification made: inserted sleeps in order to ensure the robot spent adequate time in the maze, and so that the robot could respond in time to lamp detection ~ close to 120s total.

If it was the case that the maze lasted >120s, the safety core would cut off the maze navigation and send the robot into end-phase activity. The mission takes under 500s total.

Issues Encountered

> External Task & Speed Issue

When we sought to have enough external task computation, we found that we were lacking a good deal using the base MP2 code. We believed that we had 2 options in terms of approaching this: (1) increase the period of threads in order to give more CPU time to the external tasks, or (2) decrease the speed of the robot in order to spend more raw physical time inside the maze.

Our eventual solution used neither of these - in order to give more CPU time and spend more physical time in the maze *without* massive re-tuning and overhauling, we choose to inject sleeps into the navigation thread, freezing state and sensor readings

until awoken. This leveraged our already fast maze-solving algorithm, and required no re-tuning, allowing us to focus entirely on ensuring that the external tasks were getting enough CPU time, and that we were exiting the maze at the right moments. Additionally, this gave our robot a pseudo-slowdown, allowing it to respond to the lamp more easily before moving away from it.

> *Image processing after exiting the maze*

Our Raspberry Pi non-deterministically shuts down while doing object detection after exiting the maze. We tried downsampling and segmenting images in order to minimize computational thrashing (which might be one of the problems as suggested by the TA), but this didn't solve the problem completely.