CS 424 Real-Time Systems, University of Illinois at Urbana-Champaign

Machine Problem 3

Md Tanvir Al Amin, Tarek Abdelzaher
maamin2@illinois.edu, zaher@illinois.edu

Available at https://courses.engr.illinois.edu/cs424/mp/mp3.pdf

1 Problem Description

In this MP, you will apply the principles learned in class to improve your design of the robot software (your C++ program) developed for MP2. While MP2 focused primarily on functionality needed for the robot mission (such as navigation, safety, and computer vision tasks), MP3 will focus on the overall software architecture.

We recall the mission description from MP2, which described a robotic mission to explore a newly discovered ancient underground structure (the maze). The description said: "A research team is assembled. They send a robot into the maze. The robot must map the maze while taking pictures of all encountered objects. The maze may include cliffs and traps. The robot must protect itself while in the maze. Scientists are excited and have software (payload) they want to download to the robot. This payload will execute analytics on data received. It may be buggy, but it needs to be executed in the maze because limited bandwidth prevents the robot from reporting all data outside. The robot should complete the mission as quickly as possible".

To develop a good software architecture for this mission, we must first distill the safety-critical, mission-critical, and performance/value-added requirements. Recall that *safety-critical* requirements are those that relate to basic survival/safety. Violating them can cause serious harm. They must therefore be implemented reliably and take priority over everything else. Problems with other parts of the system should not impact functionality of safety-critical components. *Mission-critical* requirements are those that must be met for the mission to succeed. They define the purpose of the mission. Violating them may cause the mission to fail (in some aspect), but the system itself will remain unharmed. Finally, *performance requirements* specify quality attributes that describe how well the mission is performed. We now list the safety-critical, mission-critical, and performance requirements for the maze exploration mission:

- **Safety-critical requirements:** Protect the robot from harm. The robot

can be harmed by collisions with walls or by dropping into a hole. Also, since the maze can be dangerous, we require in MP3 that the robot spend no more than two minutes in the maze. If the robot is in the maze for more than two minutes, a safety violation will occur.

Note on threat model:

- *Wall collisions:* In MP3, lighter walls will be set up to mimic structurally-weak ancient caves. A hard collision with a wall may topple the wall on top of the robot and physically break it - do not let that happen. Toppling a wall would be a safety violation that kills your robot (you will also be responsible for replacing any physically broken components so please test with care and be ready to catch any falling objects so they do not harm your robot). Bump sensors on the robot warn of impending collisions. When a bump sensor touches an obstacle and is depressed, you have only a small fraction of a second (depending on robot speed) before the bulk of the body of the robot hits the obstacle. You will need to bring the robot to a full stop before its core hits the wall and potentially topples it causing a safety violation.
- *Holes and cliffs:* In addition, ramps will be set up leading to cliffs. The cliff and wheel-drop sensors warn about approaching drop-offs. Should you fall off a cliff, the robot can be damaged too, so it is a safety violation.
- *Timing constraint:* To guard against other unknown threats, as mentioned above, we limit the mission time to two minutes. Failure to meet this timing constraint is a safety violation. If you are getting close to this deadline, the robot might want to reprioritize things accordingly so that it is able to get out on time. You can also choose to abort. (See Section 2 on how to safely abort a mission.)

- **Mission-critical requirements:** The mission is to do three things:
  - *Map the maze:* The robot must generate an accurate map of the layout of the maze. In MP3, the corners can be arbitrary angles, although the wall segements will still be straight (no curved walls).
  - *Identify encountered objects:* The robot must document (i.e., record and identify) all encountered objects. The list of encountered objects will be needed by the scientists to plan further excavation missions. Failures to document an object (a false-negative) and object misclassifications will be considered as violations of this mission requirement.
  - *Disarm Aladdin's Lamp:* In MP3, your mission will also include disarming Aladdin's Lamp, if it is found. This Lamp is a dangerous supernatural device feared to be in the maze. Figure 1 shows it. To disarm the Lamp, the robot must activate the Proton Transfunctioner;

Figure 1: Aladdin's Lamp

a weapon that it carries on board. To be effective, it has to be activated when the robot is near the Lamp. (Since the real robot does not actually carry weapons, you will emulate weapon activation by turning on the red light on for 2 seconds when you see the Lamp.) The transfunctioner is the only weapon that can be used to disarm the Lamp, but it has a single charge, so if you use it at the wrong time (i.e., turn the red light on when the Lamp is not in view), you will no longer be able to disarm the Lamp and will fail to satisfy this mission requirement.

  - *Run scientists' payload:* The scientists need to run some proprietory code/task during the mission, *while the robot is in the maze*.

- **Performance requirements:** The following dimensions of performance will be used to evaluate how well the mission is done:

  - *Time to traverse maze:* How long did the robot spend in the maze. The shorter the better. Note that, if you spend more than two minutes in the maze, it will be a safety violation.
  - *Time to finish entire mission:* In some designs, part of the needed processing can be postponed until after the robot gets out of the maze (and hence, such processing is not subject to the two minute deadline). The mission is done when all processing is finished. The sooner it is finished the better.

2 Safe Mission Abort

While we encourage you to do every possible effort to finish the mission successfully on time by traversing the entire maze in less than two minutes (and meet all requirements), if all else fails and you are running out of time (i.e., approaching the end of the two-minute constraint), you can engage the safe mission abort procedure. For purposes of this MP, safe mission abort is to stop the robot and turn the red LED light on as a distress signal. If you execute

safe abort before the two-minute deadline expires, you will be considered to have met the safety-critical time constraint. You will not be allowed to continue navigation after aborting. However, you will be allowed to finish any pending computation. Once you abort safely, you will be allowed to either count this mission (for grading purposes) or restart it without penalty (i.e., forget it happenned and start over). You can choose to abort a mission for other reasons as well (e.g., it is not going well). The only condition is that it needs to be done in software (by stopping the robot and turning the red LED) and not by you asking the TA to restart. You are allowed at most two aborts. Note that, you cannot abort a mission after a safety violation has occurred. In real-life the robot will already be damaged.

3 Grading

You will be graded on the *design* plus the *execution* of the mission. The execution grade will depend on how well you met safety, mission, and performance requirements. *To emphasize the point that safety violations are really critical, safety will viewed as a multiplicative factor, multiplied by the mission execution grade*. More specifically, in MP3, your robot is given 3 "lives". Each safety violation takes one of your lives, voids and restarts the mission, and reduces the multiplicative factor by 0.33 (i.e., if your robot dies three times, you get zero on mission execution). Missions you aborted safely and chose to restart are not counted. (It is as if they never happenned.) The following table shows the complete distribution of points.

Requirement	**Points**
Robot Safety	F = 1, 0.67, 0.33, or 0
Mapping Contour	15%
Object Identification	20%
Lamp Disarming	10%
Execution of Scientists Payload	10%
Time in Maze	10%
Mission Total Time	10%
Total (Execution)	**75% * F**
Design Report	15%
Demo Q&A	10%
Total	**100%**

4 Maze Environment

- Wooden logs placed inside the lab will be used as walls to create mazes that your robot needs to traverse. Only one layer of log will be used to create the walls. They might be fragile and fall if you bump too hard.

- It is possible for the walls to have different colors. For example, one wall can be white, another can be brown, and so on. Your program should be robust enough in this situation.
- The turns are not restricted to be right angles. The corners can have arbitrary angles.

5 Robot Safety

- Ensure safety of the robot at all times. You should continue to check for Cliff Signal, Overcurrent, and Wheeldrop sensors, and take appropriate actions when they trigger. In case of an actual overcurrent, you must stop the motors, play a sound, and wait for an 'advance button' input to resume mission. In case of the robot falling off a cliff or the wheels being dropped, the robot should stop the motors, play a sound, and automatically resume once the problems are solved.
- Pressing the 'play button' anytime should always stop the robot as if the traversal is complete.
- If the robot bumps a wall, your program should be responsive enough to stop or move back from the wall quickly so that the wall doesn't fall as a result of the bump. *Note that the robot must not cause a wall to fall*. Causing a wall to fall is considered a safety violation.
- If the robot bumps a wall, the vibration to the wall should be as low as possible. In other words, try to make the bumps light.

You are free to employ techniques like reducing speed while a bump could be imminent, or detecting bumps fast enough, or using the wall signals to stay safe from bumps to reduce the impact of bumps.

6 Maze Traversal

The robot should traverse the maze by closely following the wall. You should minimize bumps to the wall because they are a major factor impacting your speed and safety. The approximate length you need to traverse will be around 25 feets. It may include 6-7 corners of random angles. Your program should not take more than two minutes for the traversal. You can adjust your speed accordingly to complete the traversal within two minutes. If you miss this deadline, it is considered a safety violation.

7 Plotting Contour

Note that the corners can now have arbitrary angles. You need to plot the contour of the maze appropriately using OpenCV.

8 Object Identification

You need to identify the objects placed inside the maze. Use the camera to take photos when the robot is traversing the maze, and use OpenCV to identify the known objects. The database of known objects remains the same as mp2.

9 Disarming Aladdin's Lamp

If you encounter this lamp, you will need to turn the red LED for 2 seconds to "disarm" it. You will need to be near the Lamp for the disarming to be successful. For the purposes of this MP, if the robot can see the lamp then it is near the lamp. Once the red LED is turned on for disarming, it cannot be used again (i.e., only the first time will count), because you have only one "charge" of the disarming weapon.

10 External Tasks

We will provide an external task (as a binary executable) comprising the scientists' payload for you to run. The external task should be treated as untrusted in that it may be buggy, and cause unwanted behavior. The task may fork children. They will execute arbitrary scientific computation. The robot program can use `fork` and `exec` family to start the task, and should set its priority and permissions as appropriate for untrusted code, using appropriate system calls (as in the CS 241 course). *You must ensure safety, mission, and performance, while running these external tasks*. They will include counters to measure progress. These counters will be used for grading to assess whether you actually ran those tasks. (See mission-critical requirements.)

11 System Architecture

You must use multiple threads, and assign appropriate priorities to those. You should use an appropriate architecture of dependencies that allows the mission to complete while ensuring safety, mission, and performance requirements.

12 Project Report

Update your project report from MP2 to reflect the latest architecture with figures. The updated report should contain a diagram showing all components of the system. Please use different colors for components as follows: (i) safety-critical in red, (ii) mission-critical in blue, and (iii) performance-related in black. If a component serves more than one function (e.g., it is needed for both safety-critical and mission-critical functions), choose the color that reflects the higher

criticality. Remember that safety is more critical than mission, and mission is more critical than performance. The diagram should also show dependencies among your components. A red solid arrow from component A to component B should mean that B depends on A (i.e., failure in A will result in failure in B). A dashed black arrow from A to B means that B uses results of A. Explain in a table, which thread/component relates to which requirement, and explain their priority levels. The table should have the following three columns: (i) requirement (for example, "Disarm Aladdin's Lamp"), (ii) component name and function (for example, "The Weapon Release Thread: It is invoked when the Lamp is detected and it turns the red LED on"), and (iii) criticality level (which is one of "Safety-critical", "Mission-critical", or "Performance/Value-added"). The report should also include the algorithms used for navigating the robot. Explain the major issues encountered while solving the problem, and how you solved those. Report the maximum speed you could run your robot during your tests. Include *three different examples* of *contours plotted* using your program, *the locations where you tested it*, and *the time it required for that experiment*. Note that you are allowed to reuse any parts of the report from MP2 if those parts have not changed (e.g., algorithms that were not modified), and of course you should update any parts that changed. *The report must be in pdf format*.

13 Submission

Submission will be done via `UofI Box`. Create a folder `mp3-robotpiN` where `N` is your group number, and copy all your files including the report inside it. Upload the `mp3-robotpiN` folder to the box folder shared to your group. You will submit your final code by the deadline, and later come for a demo. You need to bring your own equipment for the demo. No code change is allowed after the deadline has passed. Therefore, during the demo, you will run the code that was submitted earlier. Be prepared to explain your work and answer questions during the demo.