DEPARTMENT OF
Systems and Computer
Engineering

Project Progress Report - SYSC 4805

Team Blue Jeans
Angela Byun *101004764*
Tarun Kalikivaya *101056745*
Martin Klamrowski *101009909*
Matthew Wesley-James *100944007*

1324 Words

Keywords: Mobile robot, maze, path-finding 11th March, 2021

Abstract
Team Blue Jeans proposes a mobile robot with maze navigating capabilities. The robot will enter a maze with the purpose of locating an objective. The robot must locate the objective, load it into its carriage, and exit the maze within the allowable time. This robot would have applications in search and rescue, and exploration. The physical elements of the robot will be simulated in CoppeliaSim. The robot's underlying intelligence will be implemented in Python and integrated with CoppeliaSim using the provided CoppeliaSim Python API.

Contents

1 Charter

1.1 Objective

The objective of this project is to design a robot that will enter a maze to find an objective then retrieve it by carrying it out of the maze. The concept is visualized in Fig. 1 below.

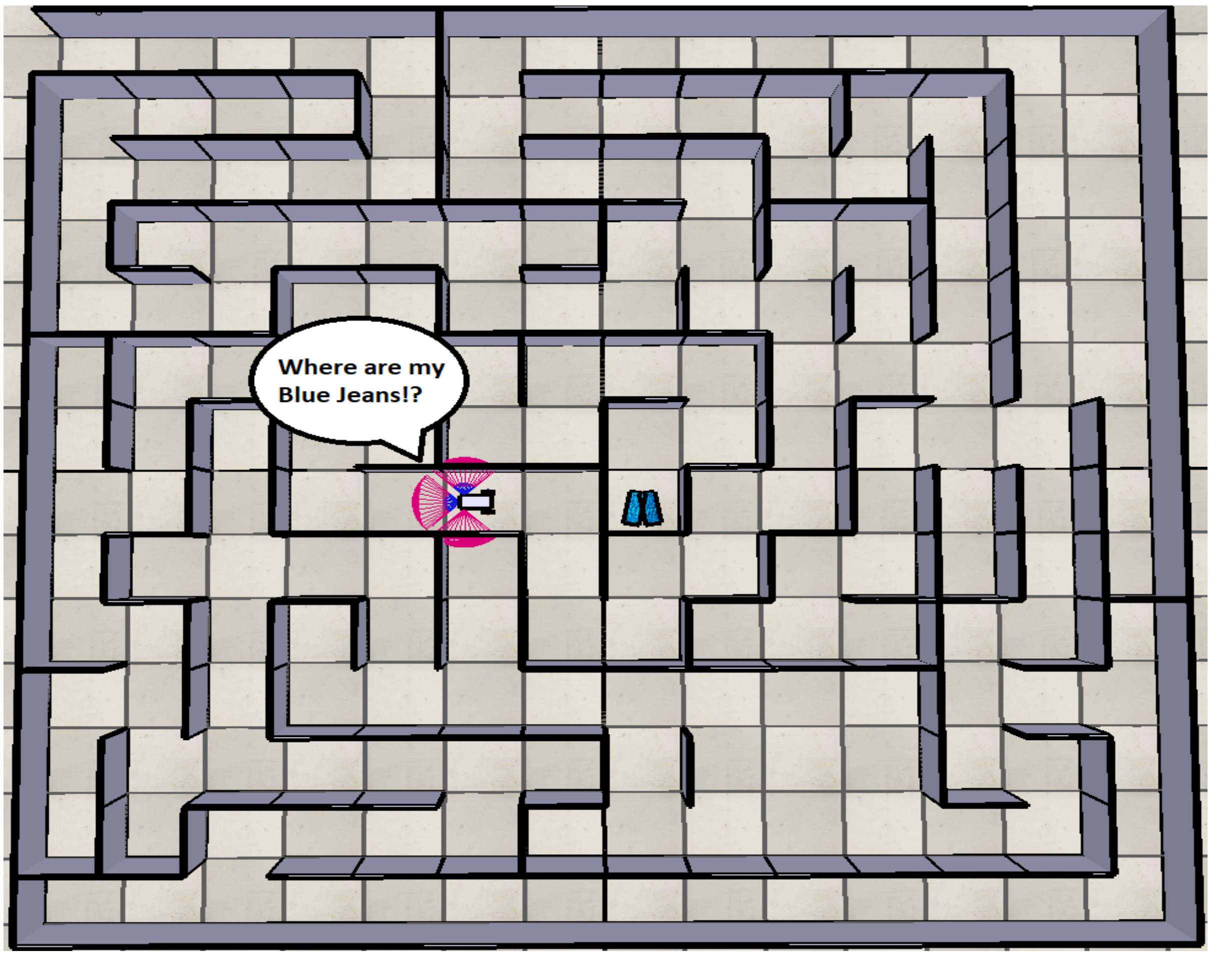


Fig. 1. Project Concept.

1.2 Deliverables

The purpose of this project is to attain two deliverables; first being the project management based on expectations for this course and the second being the final robot system, which will be the maze robot. Project management will be applied through the proposal, progress report, presentation, demonstration, and the final report by monitoring the progress. The proposal will consist of how we plan on achieving the project goal, and the basic layout of the maze and the robot, including the milestone. The maze robot system will consist of the maze, the robot, and the objective. The robot will be programmed to find its way by using different sensors, and a path finding algorithm. We believe that our challenge would be programming the robot because it will be required to move through and detect possible paths in the maze, find and carry the objective then find its way back out of the maze again.

2 Scope

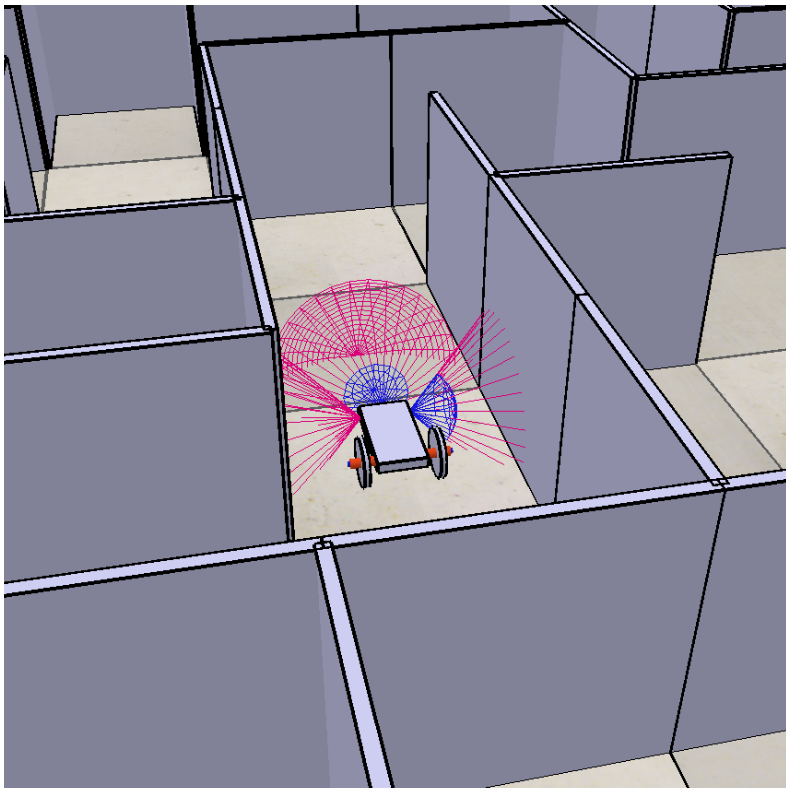


Fig. 2. Robot navigating a maze corridor.

2.1 Requirements

1. The robot must be able to maintain constant and smooth forward and backwards motion at a speed that ensures reqs. 2, 3, 4 are also met.
2. The robot must be able to perform turns up to 180° in sharpness.
3. The robot's forward sensors must detect walls in the robot's path at least 0.5m away (the length of a single wall segment).
4. The robot's lateral sensors must detect walls immediately adjacent and parallel to the robot's path at least 0.25m away (half the length of a single wall segment).
   4.1. The robot must maintain 0.25m of distance from any detected wall segments, unless doing so conflicts with reqs. 5.1, 5.2.
5. The robot's sensors must be able to detect the objective in a 180° forward-facing arc of radius 0.5m.
   5.1. When the robot detects the objective, it must turn and move to the objective.
   5.2. The robot must stop when less than 0.05m away and facing the objective.
6. The robot must move forward until a blocking wall segment is detected or req. 5.1 and 5.2 are met.

6.1. When the robot's sensors detect a wall segment in the robot's path, and only the robot's right lateral sensor detects a wall segment, the robot must perform a 90° turn to the left.

6.2. When the robot's sensors detect a wall segment in the robot's path, and only the robot's left lateral sensor detects a wall segment, The Robot must perform a 90° turn to the right.

6.3. When the robot's sensors detect a wall segment in the robot's path, and both the left and right lateral sensors of the robot detect a wall segment, the robot must perform a 180° turn in place.

7. The robot must keep track of the path it takes through the maze.

7.1. The robot must know what direction (in °) it is heading.

7.2. The robot must have an odometer to track the distance it travels in each direction.

7.3. The robot must prioritize turns into unexplored areas higher than turns into explored areas.

7.4. When presented with multiple turns of differing priorities, the robot must take the higher priority turn.

7.5. When presented with multiple turns of the same priority, the robot must randomly choose a turn to take.

7.6. The robot must take the shortest path it knows when exiting the maze (seen in Fig. 3).

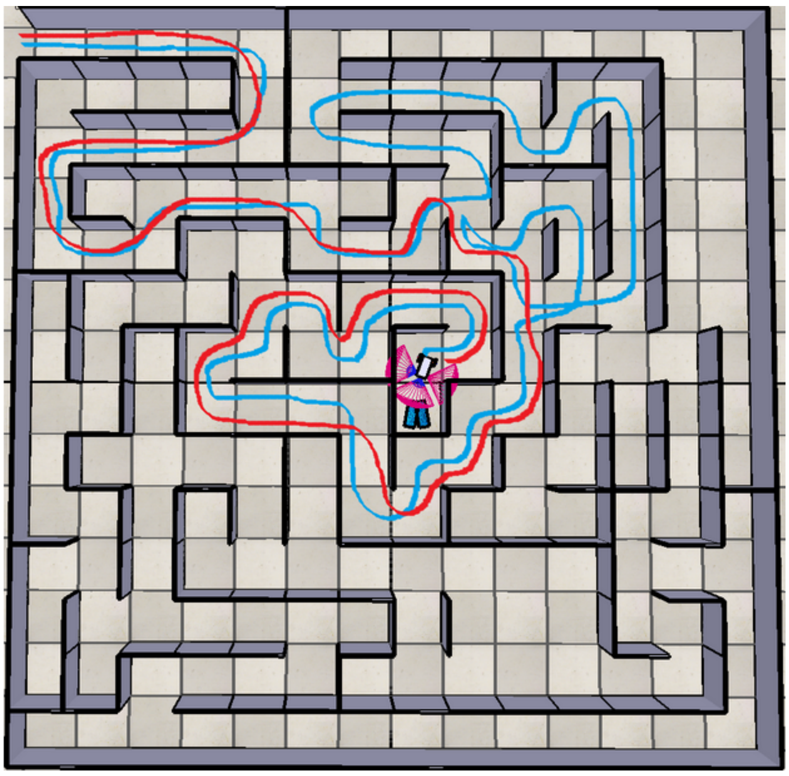


Fig. 3. Path taken by the robot in (Blue) and out (Red) of the maze.

8. The robot must have a pair of prongs at its front that can be rotated around a central axis.
    8.1. If the prongs are at 0° when parallel with the ground, the prongs must be able to rotate to an angle of 75° above the ground.
    8.2. The robot will line up the prongs with the holes of the objective when req. 5.2 is active (seen in Fig. 4).

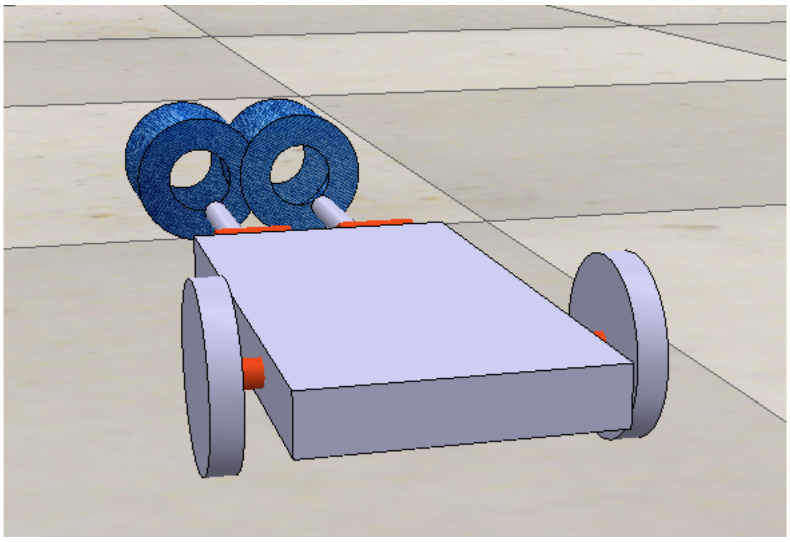


Fig. 4. The robot lining up prongs with objective holes.

    8.3. Once req. 8.2 is satisfied, the robot will secure the objective by lifting it up off the ground (seen in Fig. 5).

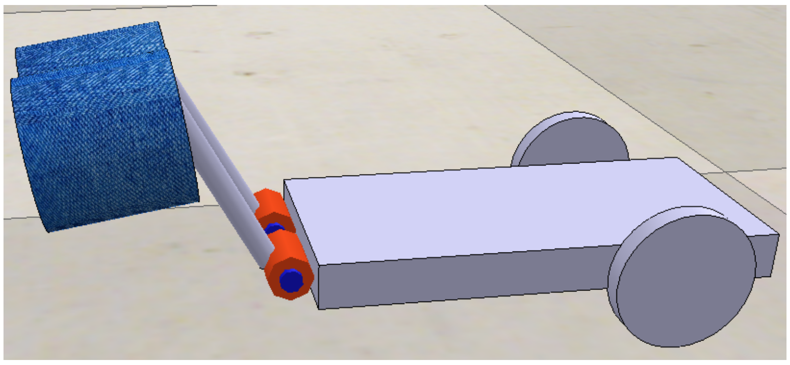


Fig. 5. The robot lifting the objective off the ground.

    8.4. Once req. 8.3 is satisfied, the robot must work to complete req. 7.6 while carrying the objective.

2.2 Event vs. Time Triggering

The various subroutines and components that make up the robot are, or will be, event-triggered.

1. Sensors:
   (a) The various sensors on the robot will trigger on specific events. For example: the robot will compute a change of course if a wall is detected (event) in its path.
   (b) Objective securing mode (req. 8.2) will be activated upon detection of the objective.
2. Navigation:
   (a) The underlying assumption in the navigation component is that the robot will be able to measure the distance it travels. Distance will be computed by implementing an odometer which will measure how many times the wheels spin.
   (b) Directions to take are determined by considering sensor readings (events) and the state of the robot's internal map (updated through events).
3. Prongs:
   (a) The objective will be considered secure when the prongs reach an angle of 75° relative to the ground. This will be ensured by setting the prongs' target joint position when req. 8.2 is satisfied.

2.3 Work Breakdown Schedule (WBS)

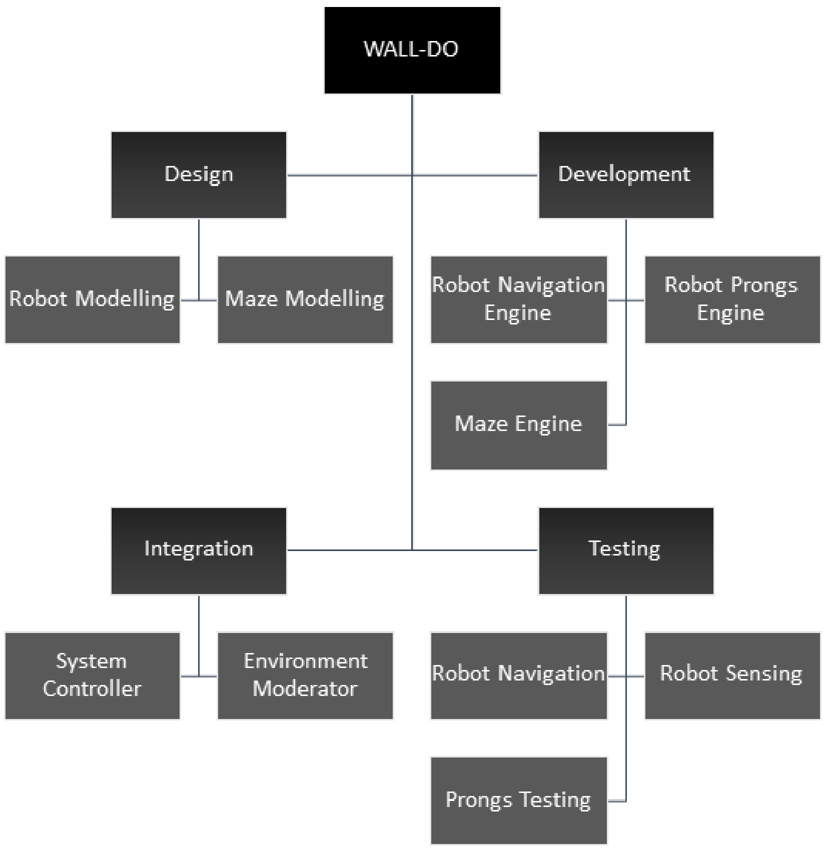


Fig. 6. WBS Diagram

2.4 Testing

- **Unit Testing**
  - Robot Navigation
    - Robot will be tested to move straight at a constant speed, turn 90°, and 180°, and stop. This will be to confirm requirements 1 and 2. These will be verified by calling getObjectPosition() in a loop.
  - Robot Sensors
    - Each sensor will be tested by having the sensor move toward a flat wall, the position of the sensor will be tracked to verify the condition is met. This will confirm requirements 3 and 4 (using get/setObjectPosition).
    - The sensor will be placed with a wall on one side and the objective on the other side (0.5m away), it will rotate in place and detect what is in front of it. This will confirm requirement 5 (using get/setObjectPosition).
  - Hook Testing
    - To complete requirements 8 (and the subsequent requirements), the hook will be activated (after requirement 5.2 is met) then the hook will be activated. Once the hook has signaled it is finished. The robot will move forward. If the robot's velocity equals the objective's velocity the requirements will be satisfied.
- **Integration Testing**
  - The robot will be placed facing towards a wall, and will run forward, if it stops 0.5m away from the wall requirement 6 is met.
  - For requirements 6.1, 6.2, and 6.3 the robot will be placed in the center with walls in four different cases (as shown in Fig. 7 below.) Once the correct direction is chosen the requirement will be met.
  - For case A the robot will be pre-programmed with each of the three directions having different unexplored area values, this will test the requirements 7.4, 7.5, and 7.6.

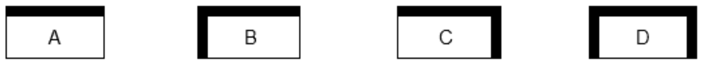


Fig. 7. A: Front-facing wall. B: Front-facing wall and adjoining left wall. C: Front-facing wall and adjoining right wall. D: Dead end.

- **System Controller Testing**
  - For the requirements of 7 (and the subsequent requirements) a custom made testing maze will be made, with trackers for each requirement. These will be waiting until the robot goes to a certain positions that will be calculated based on the custom testing maze, then these requirements will be complete.

- **Full System Testing**
  - Robot will enter the maze (assuming robot moves at 0.5m per second) the robot will have to exit the maze within the time limit to be successful. The time limit in seconds t_{max} is determined by Eq. (1), where l_M, h_M represent the maze length and height respectively in meters.

$$t_{max} = 2l_M h_M + 5 \tag{1}$$

3 Schedule

3.1 Schedule Network Diagram

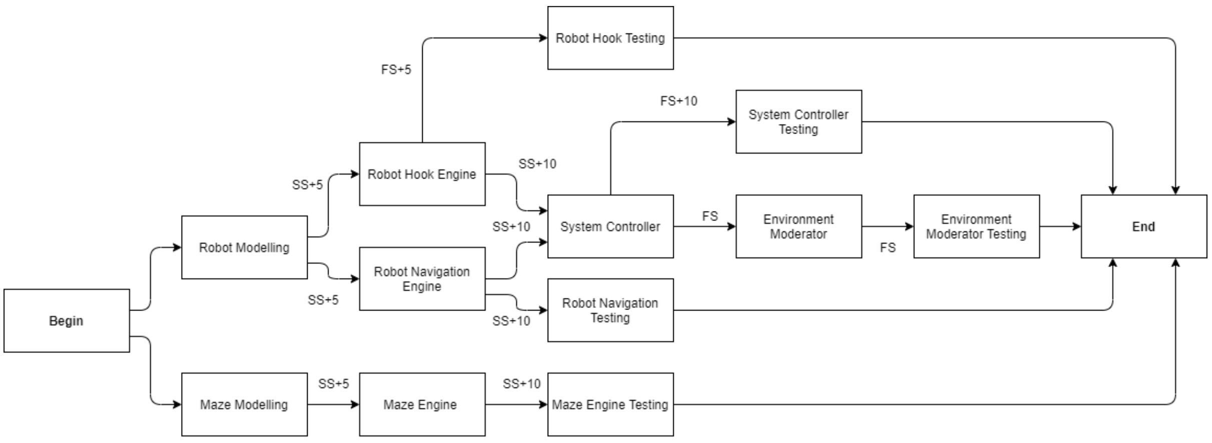
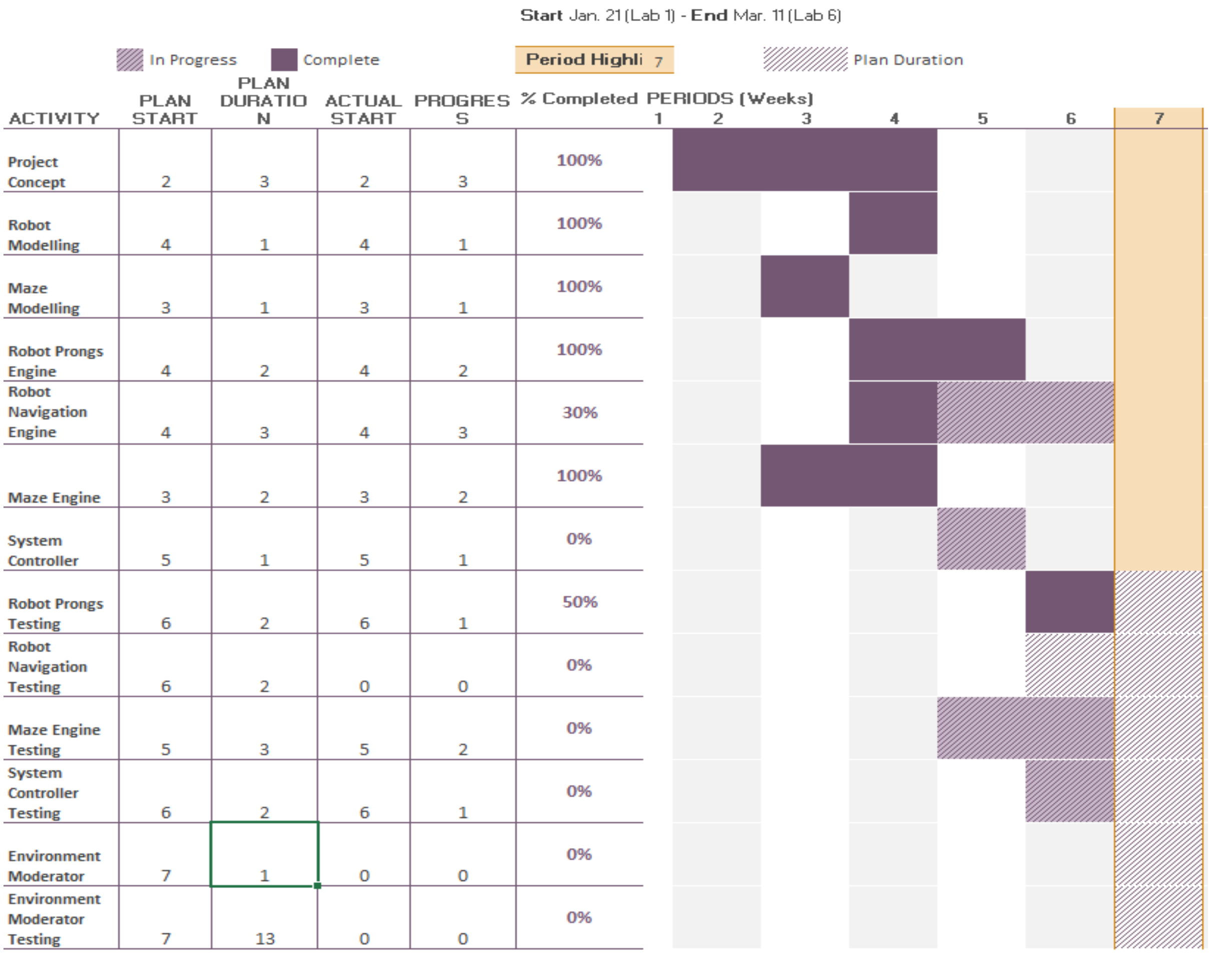


Fig. 8. Schedule Network Diagram

3.2 Gantt Chart



Start Jan. 21 (Lab 1) - **End** Mar. 11 (Lab 6)

▨ In Progress ■ Complete **Period Highli** 7 ░ Plan Duration

ACTIVITY	PLAN START	PLAN DURATIO N	ACTUAL START	PROGRES S	% Completed	PERIODS (Weeks) 1	2	3	4	5	6	7
Project Concept	2	3	2	3	100%		■	■	■			
Robot Modelling	4	1	4	1	100%				■			
Maze Modelling	3	1	3	1	100%			■				
Robot Prongs Engine	4	2	4	2	100%				■	■		
Robot Navigation Engine	4	3	4	3	30%				■	▨	▨	
Maze Engine	3	2	3	2	100%			■	■			
System Controller	5	1	5	1	0%					▨		
Robot Prongs Testing	6	2	6	1	50%						■	░
Robot Navigation Testing	6	2	0	0	0%						░	░
Maze Engine Testing	5	3	5	2	0%					▨	▨	░
System Controller Testing	6	2	6	1	0%						▨	░
Environment Moderator	7	1	0	0	0%							░
Environment Moderator Testing	7	13	0	0	0%							░

Fig. 9. Gantt Chart for Activities

4 Diagrams

4.1 Architecture

The system architecture is visualized in Fig. 10 below. Only the ”physical” robot is in CoppeliaSim. The underlying motor function, navigation logic, object detection is done in Python, with a connection being established with CoppeliaSim through the provided RemoteAPI module. The details for communicating and needed function calls are abstracted away in the Boundary class.

The Robo class maintains an instance of Boundary. The Robo class will define the robot’s underlying behaviour. Robo also maintains an instance of MazeMap, which is used to model the maze the robot is exploring. MazeMap is essentially a 2D array of MapNodes. MapNodes are essentially cells with variable surrounding walls.

Robo also makes use of Maze, which contains algorithms for generating and solving a maze.

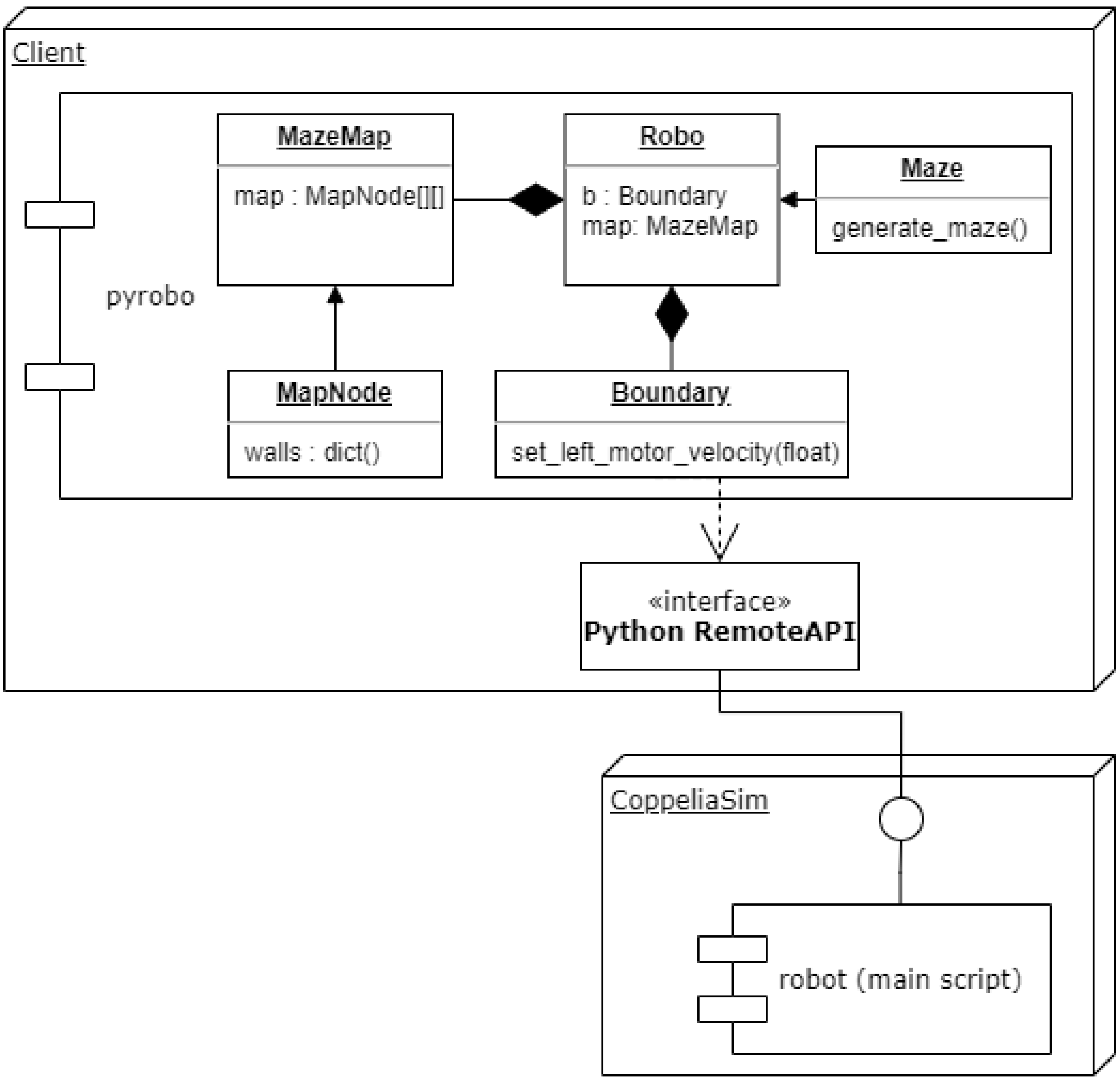


Fig. 10. System Architecture

4.2 State Chart Diagram

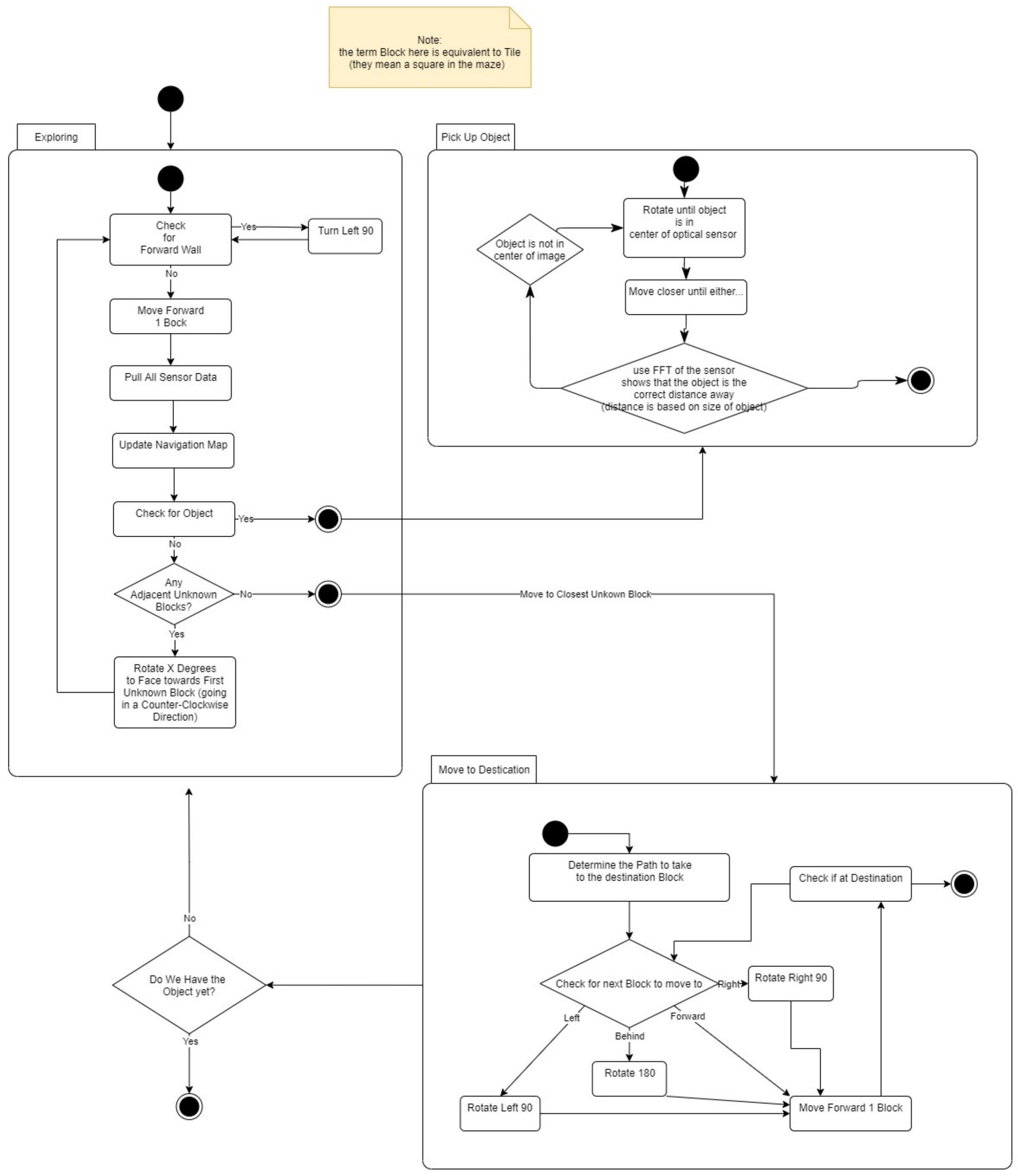

Note:
the term Block here is equivalent to Tile
(they mean a square in the maze)
Exploring
Check for Forward Wall
Yes
Turn Left 90
No
Move Forward 1 Bock
Pull All Sensor Data
Update Navigation Map
Check for Object
Yes
No
Any Adjacent Unknown Blocks?
No
Yes
Rotate X Degrees to Face towards First Unknown Block (going in a Counter-Clockwise Direction)
Move to Closest Unkown Block
Pick Up Object
Rotate until object is in center of optical sensor
Object is not in center of image
Move closer until either...
use FFT of the sensor shows that the object is the correct distance away (distance is based on size of object)
Move to Destication
Determine the Path to take to the destination Block
Check if at Destination
Check for next Block to move to
Right
Rotate Right 90
Left
Behind
Forward
Rotate 180
Rotate Left 90
Move Forward 1 Block
Do We Have the Object yet?
No
Yes

Fig. 11. State Chart Diagram

4.3 Sequence Diagram

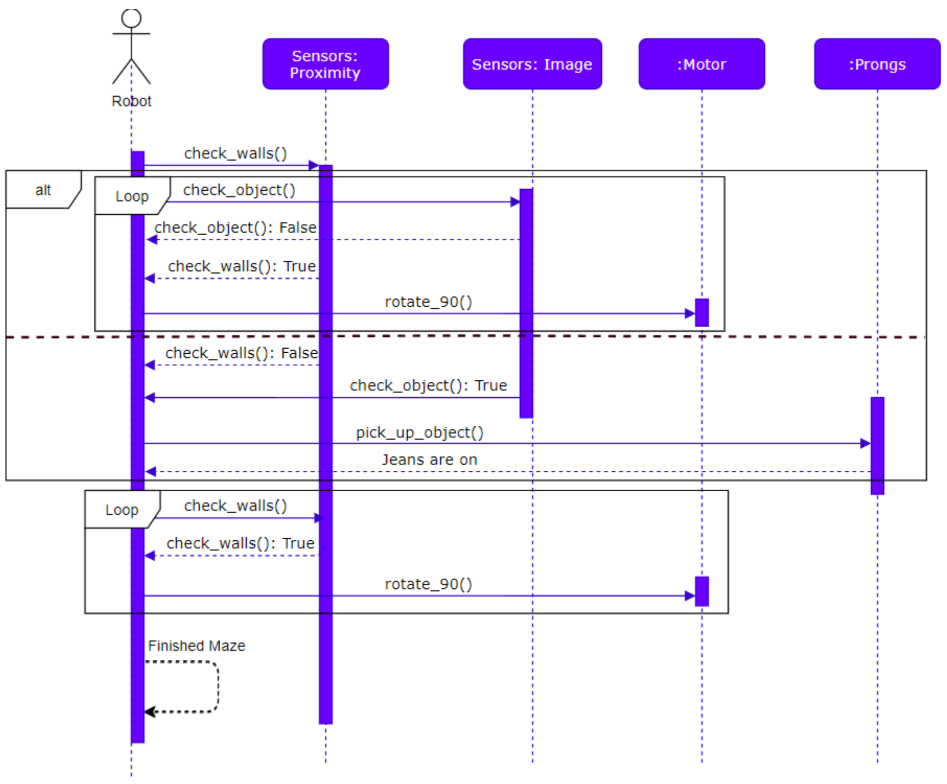
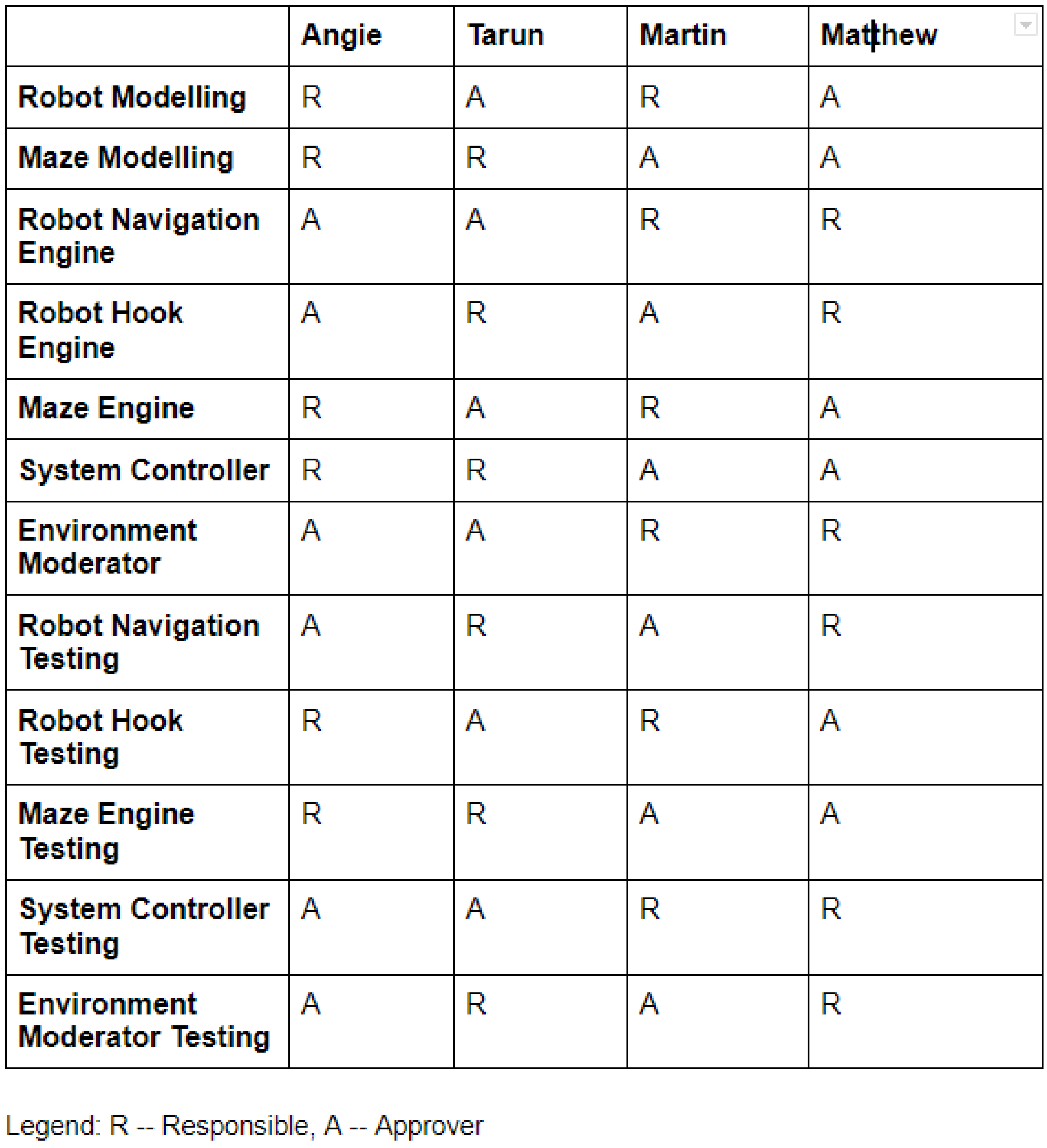


Fig. 12. Sequence Diagram

5 Human Resources (HR)

5.1 Responsibility Assignment Matrix



	Angie	Tarun	Martin	Matthew
Robot Modelling	R	A	R	A
Maze Modelling	R	R	A	A
Robot Navigation Engine	A	A	R	R
Robot Hook Engine	A	R	A	R
Maze Engine	R	A	R	A
System Controller	R	R	A	A
Environment Moderator	A	A	R	R
Robot Navigation Testing	A	R	A	R
Robot Hook Testing	R	A	R	A
Maze Engine Testing	R	R	A	A
System Controller Testing	A	A	R	R
Environment Moderator Testing	A	R	A	R

Legend: R -- Responsible, A -- Approver

Fig. 13. Responsibilities.