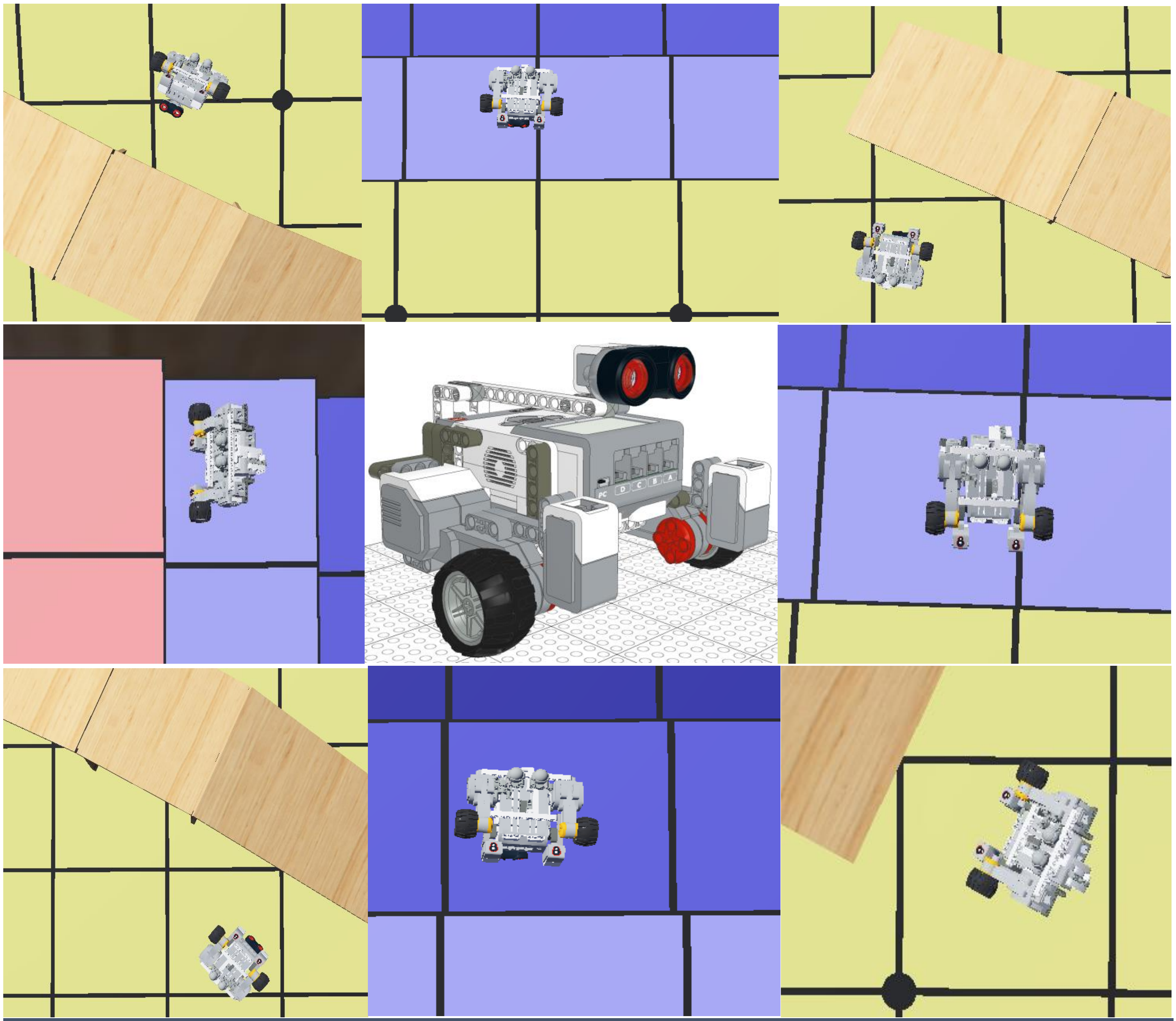


Team 1 RobotExpress

Software Document

Yutong Wang, Shichang Zhang, Junjian Chen, Linpei Duan, Lide Cui, Dominic Chan

ECSE 211

SOFTWARE DESIGN DOCUMENT

Project: Obstacle track-racer – A competitive robot design project
Task: Design and implement a working model out of EV3 Lego sets that would autonomously navigate to a racetrack on an island shown in the world map and complete as many laps as possible within the 5-minute time limit, eventually returning to its starting point.
Document Version: 4.06
Date: 11/4/2021
Author: Lide Cui, Angelina Duan, Junjian Chen, Shichang Zhang
Edit History:

Version 1.01	25/02/2021	Lide Cui	Work on the initial document.
Version 1.02	03/03/2021	Dominic Chan	Formatting and Template of document
Version 2.01	05/03/2021	Lide Cui	Detailed design of avoiding obstacles
Version 2.02	12/03/2021	Lide Cui	Add state machine
Version 2.03	13/03/2021	Lide Cui	finish obstacle avoidance, navigation, review abstract
Version 3.01	23/03/2021	Angelina Duan	initial version of thread layout and update class hierarchy and flowchart
Version 3.02	28/03/2021	Angelina Duan	update flowcharts, sequence diagram, delete state machine
Version 3.03	28/03/2021	Angelina Duan, Lide Cui, Junjian Chen, Shichang Zhang	Add software version update table, software version detail with test links and the discussion on beta demo
Version 4.01	4/4/2021	Junjian Chen, Shichang Zhang	Add update of software version 1.8
Version 4.02	5/4/2021	Junjian Chen, Shichang Zhang	Revise bridge passer and path manager. Add update of software version 1.9, 2.0
Version 4.03	6/4/2021	Junjian Chen, Shichang Zhang	Add Section 5.7:Overpass Update Section 5.4 and 5.5
Version 4.04	9/4/2021	Shichang Zhang	Add update of software version 2.1
Version 4.05	10/4/2021	Junjian Chen Shichang Zhang	Add update of software version 2.2
Version 4.06	11/4/2021	Junjian Chen, Shichang Zhang	Add update of software version 2.3

1.0 TABLE OF CONTENTS

2.0 ABSTRACT 2

3.0 SOFTWARE FLOWCHART 2

4.0 CLASS HIERARCHY 4

5.0 DESIGN OF DIFFERENT FUNCTIONS 4

5.1 LOCALIZATION 4

5.2 NAVIGATION 6

5.3 OBSTACLE AVOIDANCE 7

5.4 BRIDGE PASSER 8

5.5 PATH MANAGER **8**
5.6 WIFI CLASS **9**
5.7 Overpass **9**
6.0 THREAD LAYOUT **10**
7.0 SOFTWARE VERSION UPDATING TABLE **10**
7.1 SOFTWARE VERSION DETAIL WITH TESTS ASSOCIATED **11**
8.0 DISCUSSION ON BETA DEMO **20**

2.0 ABSTRACT

In this software document, the gameplay **flowchart** and **sequence diagram** describe the sequential behavior of the robot. All implemented methods are described respectively with a brief description. The algorithm of all main methods is illustrated in a flowchart form. A **class diagram** is also included to showcase the structure of the software for this project. Lastly, this document contains an **updating table** to demonstrate the evolution of the robot's software as well as the link to the testing doc and detailed information **after it.**
Also, a **beta demo review**, analysis and discussion are also included.
Note: Click the **bold mark** above will directly guide you to the sections.

3.0 SOFTWARE FLOWCHART

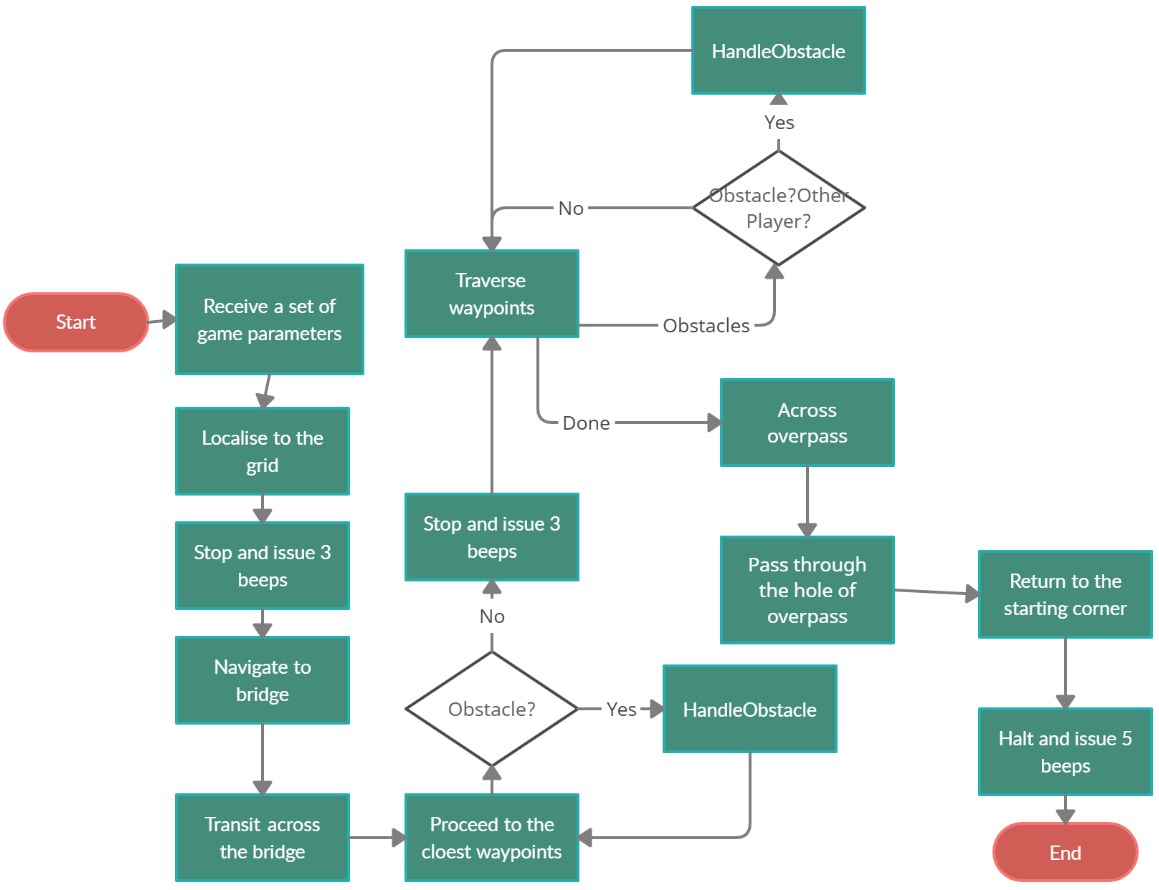


Figure 1 Major Flowchart
We construct the above major flowchart to show basic behaviors of the robot. To make the flowchart clear and in detail, we construct some other flowcharts in section 5.0 to support its functionalities.

First of all, we list the major tasks of the robot:

1. Receive parameters
2. Localization
3. Navigation to bridge
4. Across the bridge
5. Manage the path
6. Traverse through waypoints
7. Across the overpass
8. Pass through the hole of the overpass
9. Navigation back

From the major flowchart, we can see the major tasks are mostly related to localization, navigation, obstacle avoidance, bridge passer, path manager, and wifi class. To make the relationship between classes and tasks clear we construct the class diagram and thread layout in sections 4.0 and 6.0.

4.0 CLASS HIERARCHY

The following class diagram shows the relationship between each class and lists what the classes do in general. If interested, you can click this link to view the detailed class diagram.

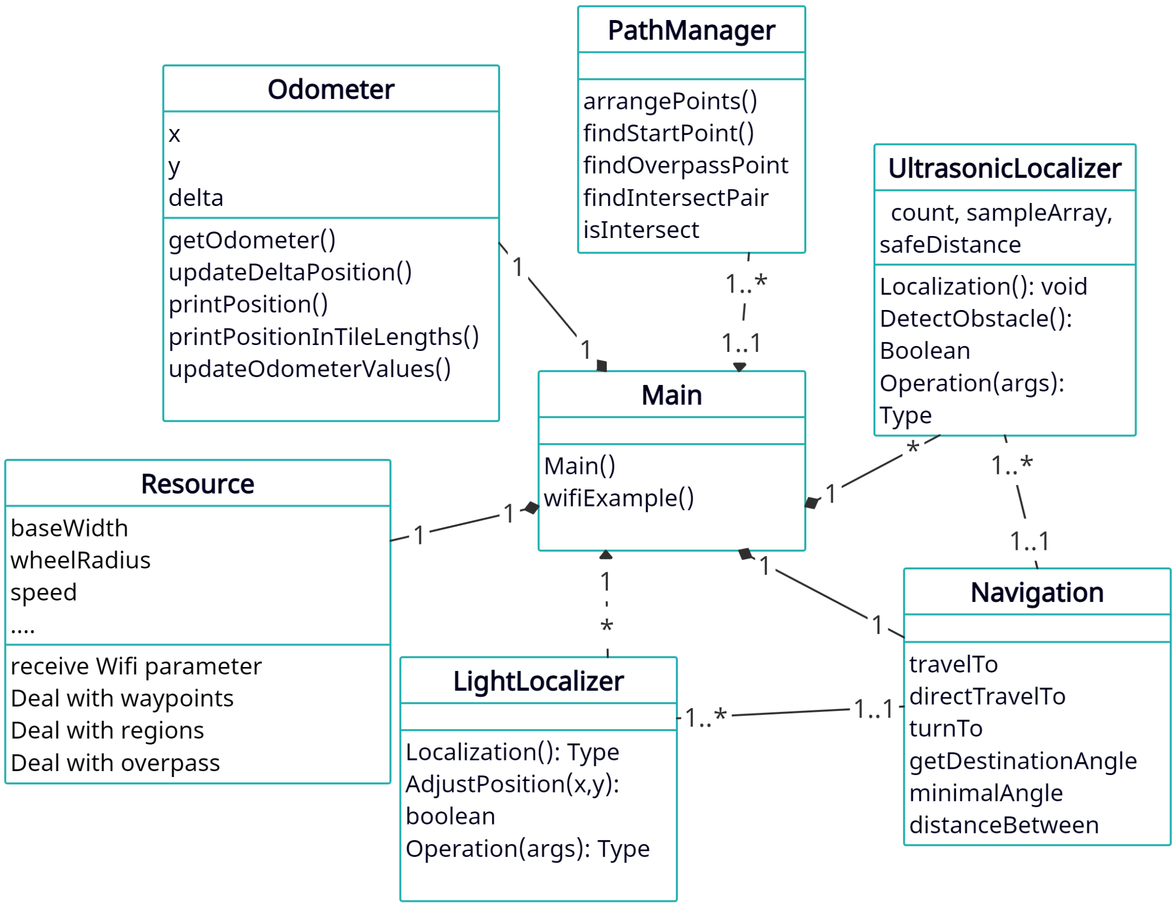


Figure 2.class diagram

Figures 3, 4, and 5 in sections 5.1 and 5.3 will show the flows for light localization, ultrasonic localization, and obstacle avoidance respectively.

5.0 DESIGN OF DIFFERENT FUNCTIONS

This section will introduce the main flow and logic of functions: localization, navigation, obstacle avoidance, bridge passer, path manager, wifi class and overpass.

5.1 LOCALIZATION

Title: Light Localization

Actor: Robot

Intention: The robot intends to localize to the start point.

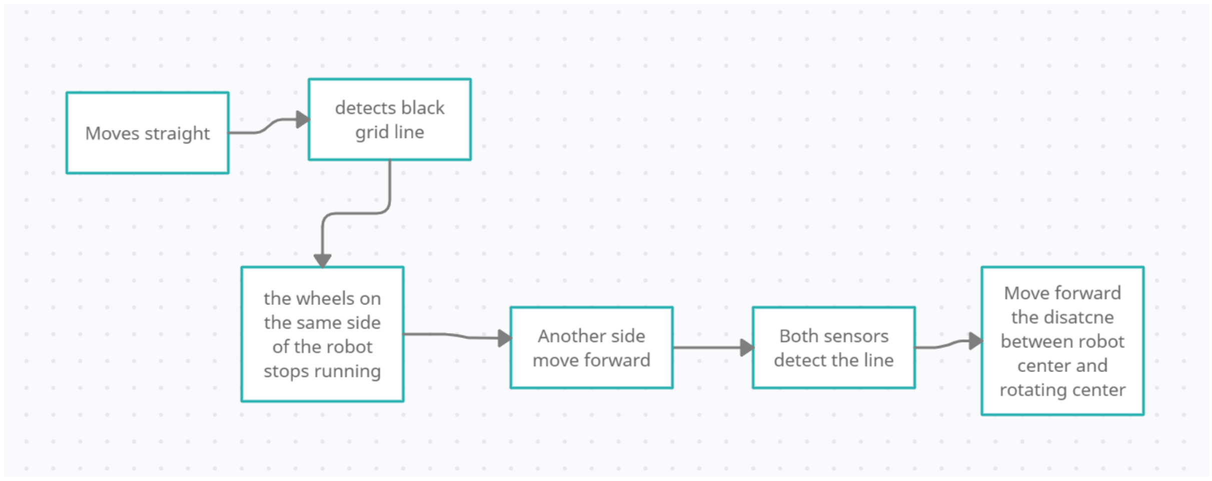
Precondition: The robot has received all parameters and finished the ultrasonic localization.

Main Scenario:

1. The robot continuously moves straight.
2. If the light sensor detects the black grid line, the wheel on the same side stops running.
3. If both light sensors successfully detect the grid line, the robot turns 90 degrees.
4. The robot continuously moves straight.

5. If the light sensor detects the black grid line, the wheel on the same side stops running.
6. If both light sensors successfully detect the grid line, the robot moves straight for a distance that is the distance between the robot center and rotating center.

Notes:
The light sensor detects the black grid line when the filtered sample reading<100.
Stopping the wheel on the same side means the left wheel stops rotating when the left light sensor detects the grid line or the right wheel stops rotating when the right light sensor detects the grid line.
The rotating center is the center of the robot wheels. The distance between the rotating center and the robot center is 0.0405m.


Moves straight
detects black grid line
the wheels on the same side of the robot stops running
Another side move forward
Both sensors detect the line
Move forward the disatcne between robot center and rotating center
Figure 3. Flowchart for initial light localization

Title: Ultrasonic Localization
Actor: Robot
Intention: The intention for the robot to face toward 90 degrees.
Precondition:
1. The angle of the odometer should be set to 0 before starting.
2. The robot should be placed at the 45-degree line of the corner.
3. A constant called “safe distance” is set (13-25)
4. A constant called “face distance” is set (50-70)

Main Scenario:
1. The robot will turn right until the ultrasonic reading is lower than the safe distance
2. Set the theta of the odometer to 0
3. The robot will turn left until the ultrasonic reading is larger than the face distance
4. Record (360-theta of odometer) as angle A, turn right by angle A
5. Set the theta of the odometer to 0
6. The robot will turn right until the ultrasonic reading is larger than the face distance
7. Record theta of odometer as angle B
8. Turn left by angle (A+B)/2
9. Turn right by 135 degrees

Alternative\Exception #1: The robot keeps turning without stopping. This is because the safe distance is too high and the reading of the ultrasonic sensor cannot go lower than it. In this case, set the safe distance higher to test.

Alternative\Exception #9: After turning, some errors may exist and the robot does not face toward 90 degrees perfectly. In this case, the light localization after it will correct the angle.
Notes:
The value of safe distance and face distance are needed to be adjusted corresponding to the position of the robot at the 45-degree line.

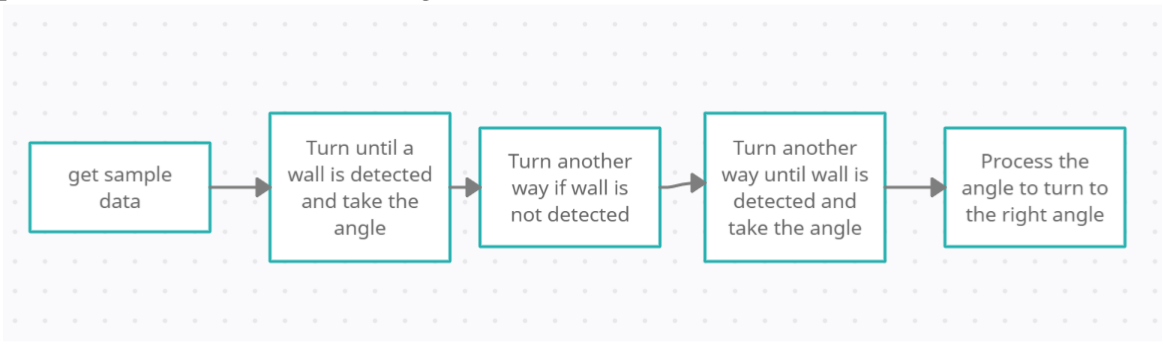

get sample data
Turn until a wall is detected and take the angle
Turn another way if wall is not detected
Turn another way until wall is detected and take the angle
Process the angle to turn to the right angle

Figure 4. Flowchart for initial ultrasonic localization

5.2 NAVIGATION

Title: General Navigation
Actor: Robot, Obstacles
Intention: The intention for the robot is to cross the bridge, run laps, and return home within 5 minutes.
Precondition: The robot has received all parameters and finished localization.
Main Scenario:
1. The robot turns to the bridge and moves to the edge of the bridge.
2. The robot moves across the bridge.
3. The robot moves the closest waypoint.
4. The robot locates itself, and if there is time left, the robot moves the next waypoint (back to step 3).
5. The robot returns to the edge of the bridge.
6. The robot across the bridge.
7. The robot moves to the starting point.

Alternative\Exception #1:
3a. If there is an obstacle, then run the obstacle avoidance process, continue at step 4.
Alternative\Exception #2:
Notes:
The edge of the bridge is the midpoint between TNR_UR_x and TNR_LL_x.
The distance of the bridge is TNR__UR_y and TNR_LL_y.
The estimated return time is 4’10’

Light localization process during the navigation:

Title: Light Localization
Actor: Robot
Intention: The intention for the robot is to localize to the asserted waypoint.
Precondition: The robot has an accurate odometer angle and it has received parameters related to travel to the asserted waypoint.
Main Scenario:
1. The robot is navigating to the asserted waypoint. It continuously moves straight along the calculated angle.

2. If the distance between the robot's current position and the waypoint is greater than the localization distance, the robot moves straight until the left distance is equal to the localization distance. If the distance between the robot's current position and the waypoint is less than the localization distance, the robot stops immediately.
3. When the robot stops, according to the odometer angle reading, knowing the robot currently orients to the 1st, or 2nd, or 3rd, 4th quadrant or on the x-axis or y-axis.
4. If the robot is pointing along the x-axis, the robot will turn to 90 or 270 degrees and move straight for a 0.5*TILE_SIZE distance and turn by 180 degrees finally. If the robot is pointing along the y-axis, the robot will turn to 0 or 180 degrees and move straight for a 0.5*TILE_SIZE distance and turn by 180 degrees finally. Otherwise, according to different quadrants, the robot just turns to 0, or 90, or 180, or 270 degrees.
5. The robot starts light localization.
6. The robot turns to the original orientation, which is determined by the last waypoint and the current waypoint.

Notes:
Parameters related to travel to the asserted waypoint are such as angles, distances.
Localization distance is 0.7ft.

5.3 OBSTACLE AVOIDANCE

Title: Avoid an Obstacle
Actor: Robot, Obstacles
Intention: The robot intends to safely avoid an obstacle by passing by it and move to the destination point.
Precondition: The robot has accurately located a point, turned by a minimal angle towards the destination point, and calculated the distance to the destination.
Main Scenario:

1. The ultrasonic sensor of the robot measures distance to the nearest 'obstacle'.
2. The robot divides the path into 3 parts.
3. The robot travels the first part (the length of the first part = distance to the obstacle – react distance).
4. The robot turns 90 degrees and detects if there is enough space to turn.
5. The robot moves the second part (the length of the second part = 3*obstacle size + 2* react distance).
6. The robot turns -90 degrees.
7. The robot moves the third part to the destination point (the length of third part = distance to destination – length of first part – obstacle size – 2 * react size).

Alternative\Exception #1:
1a. If the distance to obstacle is greater or equal to distance to destination, then the main scenario is jumped, the robot directly travels to the point.
Alternative\Exception #2:
1b. If the path of current position and destination position pass through an overpass, then the main scenario is jumped, the robot directly travels to the point.
Alternative\Exception #3:
4a. If the robot detects a wall so that it has no sufficient space to turn
4a.1. The robot turns 180 degrees.
4a.2. The robot does step 5.
4a.3. The robot turns 90 degrees and continues from step 7.
Postcondition: The robot approaches the destination point and starts the localization process.

Notes: React distance is estimated to be 0.1 m.

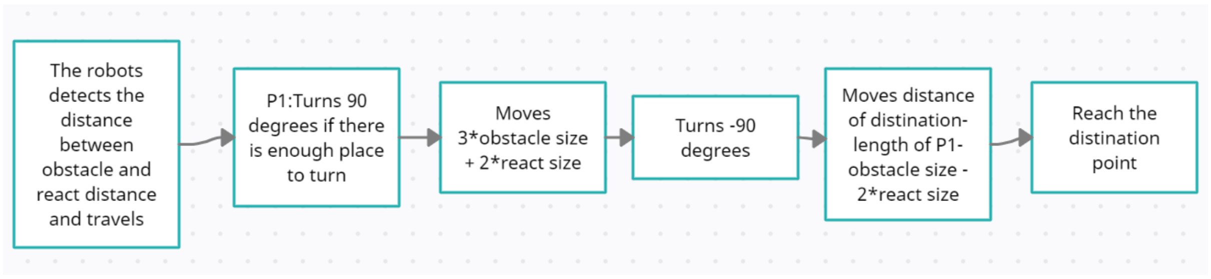

The robots detects the distance between obstacle and react distance and travels
P1:Turns 90 degrees if there is enough place to turn
Moves 3*obstacle size + 2*react size
Turns -90 degrees
Moves distance of distination-length of P1-obstacle size - 2*react size
Reach the distination point

Figure 5. Flowchart for the obstacle avoidance

5.4 BRIDGE PASSER

Title: Bridge Passer
Actor: Robot, Tunnel
Intention: The robot intends to cross the tunnel and arrive at the island safely.
Precondition: The robot has finished localizing itself and determined its team color.
Main Scenario:

1. The robot calculates the middle point 4 midpoints of the tunnel's sides.
2. The robot iterates each point. If the point is on the edge of team land, then mark it as the start point. Then it marks the point on the opposite side as the end point.
3. Set two points, pre-start and pre-end point according to the positions of start and end point.
4. The robo passes the tunnel by navigating through waypoints in this sequence: pre-start point->start point->end point->pre-end point.

Postcondition: The robot safely arrives at the island and prepares to start navigating to waypoints.

5.5 PATH MANAGER

Title: Path Manager
Actor: Robot
Intention:

1. The robot intends to rearrange the waypoints so that it can start at the closest point, find the point to go over the overpass, and find the point to go under the overpass.
2. It can calculate the points used to pass the overpass.
3. It can calculate the points used to pass the underpass.

Precondition: The robot has arrived at the island.
Main Scenario:

1. Intention 1: Traverse through every point in the given list of waypoints. Find the point which is most closed to the tunnel, set it as the first waypoint to navigate. Then rearrange the rest of points in its original sequences
2. Intention 2: Extend the line between two endpoints of the overpass and find the intersection between their nearest grids, named as "OverpassStartLocalizationPoint" and "OverpassEndLocalizationPoint". The former is used as a helper point to go before going on the overpass, the later is for localizing after going down the overpass.
3. Intention 3: Calculate the midpoint of the overpass. Calculate the normal according to the midpoint and the line made by the overpass endpoints. Pick two points on the normal that are on different sides of the overpass with an adjusted distance to the midpoint. The robot is expected to travel to the helper points on the same side of the overpass and then travel to the helper point on the other side to go through the bottom of the overpass.

Postcondition: The robot starts the navigation process.
Alternative/Exception #1:

3a. If the overpass is vertical, set the slope of the overpass to be the maximum value of Double type.

3a. If the overpass is horizontal, set the slope of the normal line to be the maximum value of Double type.

Notes:

1. The adjusted distance is 0.3048m.

5.6 WIFI CLASS

We use API from outside wifi classes. The wifi connection can be disabled to run local tests. This configuration can be set in Resources. Enable_Debug _wifi_print, and Receive_Wifi_Param. Detailed simulated world layout can be changed in lay_out

5.7 OVERPASS

Title: overpass

Actor: Robot, overpass

Intention:

1. During the navigation, the overpass can not be regarded as an obstacle and avoid it. The robot should go over it and then continue the navigation process.

Precondition: the robot knows at this waypoint it should go overpass.

Main Scenario:

1. The robot knows that it should go overpass at this point.
2. The robot navigates to the "OverpassStartLocalizationPoint" (helper point).
3. The robot turns to the angle which the overpass points to.
4. The robot keeps move straight until it detects a black line
5. The robot uses light localization to localize to "OverpassEndLocalizationPoint".
6. The robot sets the odometer to match the position of " OverpassEndLocalizationPoint" (helper point).
7. The robot navigates to the next waypoint.

Postcondition: the robot arrives at the next waypoint.

Notes:

1. The path manager passes points that need to perform overpass action to the robot.
2. The path manager will calculate the helper points for this overpass algorithm.

Title: underpass

Actor: Robot, overpass

Intention:

1. During the navigation, the overpass can not be regarded as an obstacle and avoid it. The robot should go under it and then continue the navigation process.

Precondition: the robot knows at this waypoint it should go underpass.

Main Scenario:

1. The robot then knows that it should go underpass at this point.
2. The robot is navigated to the helper point on the same side of the overpass.
3. The robot is navigated to the helper point on the other side of the overpass.
4. The robot is navigated to the next waypoint.

Postcondition: the robot arrives at the next waypoint.

Notes:

1. The path manager passes points that need to perform underpass action to the robot.
2. The path manager will calculate the helper points for this underpass algorithm.

6.0 THREAD LAYOUT

The following figure shows the thread organization of the final project:

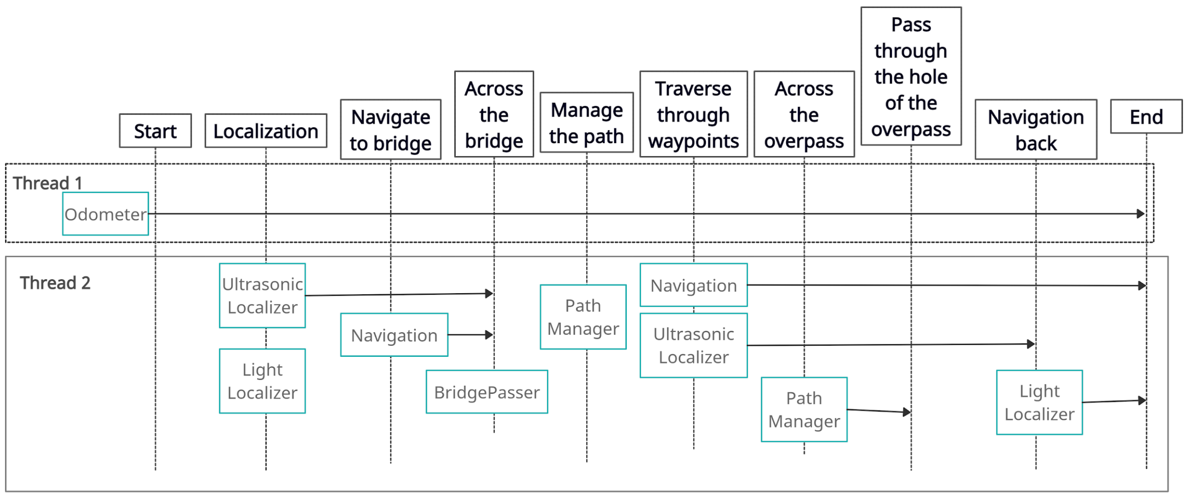


Figure 6. Sequence diagram

We separate all states into two different threads and components in threads are parallel. The first thread is used for the odometer so that we can drive the robot no matter what tasks it needs to do and they are used in many processes throughout the design. The second thread is used for localization, navigation, bridge passer, and path manager, processes inside are linked to each other.

The upper thread organization diagram has lifelines, showing the main process of the robot. When the robot receives game parameters, the localization happens at first. Then the robot knows where it is, we start the bridge localization. Once the bridge is localized, the robot crosses the bridge. Now, the path manager comes, this method finds the closest point as the start point, rearranges the waypoints list, and finds the point across the overpass. Then, it’s time to use the navigation class to traverse waypoints. During this thread, the robot can cross the overpass or pass through the hole of the overpass. Obstacle Avoidance needs to be used throughout the processes in each thread or even methods. Lastly, the robot needs to return to its starting point.

7.0 SOFTWARE VERSION UPDATING TABLE

The following table shows all the software versions with their version numbers, general description, and their commit links on GitHub.

Version number	Description	Commit links
1.0	The initial version from lab 5 included initial localization and navigation classes	V1.0
1.1	Ultrasonic and light localization tests pass	V1.1
1.2	Pass the bridge with hardcode (initial version of bridge passer)	V1.2

1.3	Can pass the overpass but with errors when facing edge cases	V1.3
1.4	Overpass tests pass	V1.4
1.5	Revise light localization, Pass the test of going under the overpass, Bridge passer updated, the initial version of return back	V1.5
1.6	Revise bridge passer	V1.6
1.7	Revise return back, add a statement to judge waypoints	V1.7
1.8	Revise Path Manager, now able to calculate the points of start and end points when going overpass and underpass	V1.8
1.9	Revise Bridge Passer, now it is able to handle cases of horizontal and vertical tunnels in both teams.	V1.9
2.0	Revise Path Manager, now it is able to calculate the overpass related points accurately.	V2.0
2.1	Revise Obstacle Avoidance	V2.1
2.2	Revise LightLocalizer and Bridge Passer. Now the robot will perform a light localization after passing the tunnel.	V2.2
2.3	Revise Obstacle Avoidance	V2.3

Table 1. Software version table

7.1 SOFTWARE VERSION DETAIL WITH TESTS ASSOCIATED

This section shows the detailed information of all software design versions with tests associated with them.

Software Design Version 1.1:

In the first version of our project software design, the ultrasonic localization of Lab5 is revised to fit the requirement of the project. In Lab5, the parameter “safe distance” is larger

than the "face distance". However, because the presence of the tunnel will disturb our ultrasonic sensor's readings, we change the setting of "safe distance" and "face distance" where safe distance is smaller than face distance now. The robot is able to use ultrasonic localization to localize the robot to face toward 180 degrees.
After that, the robot can use the light localization to localize the robot to point(1,8) perfectly. The light localization methods are the same as we used in Lab 5.

The testing on Software Design V1.1 is shown in the table below:

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:
V1.1	Ultrasonic Localization	Pass	Test 1 of Ultrasonic Localization
V1.1	Light Localization	Test 1 Pass, Test 2 Conditional pass	Test 1,2 of Light Localization testing document

Software Design Version 1.2:
At software Design 1.2, we added the simple hard code algorithm to go across the bridge and navigate to the first waypoint.
As is shown in figure 7.11, our logic is simply to go to the start point of the bridge, move straight for 3.5ft to go across the bridge. Then we use the navigation system to let the robot travel to the first waypoint (1,4).

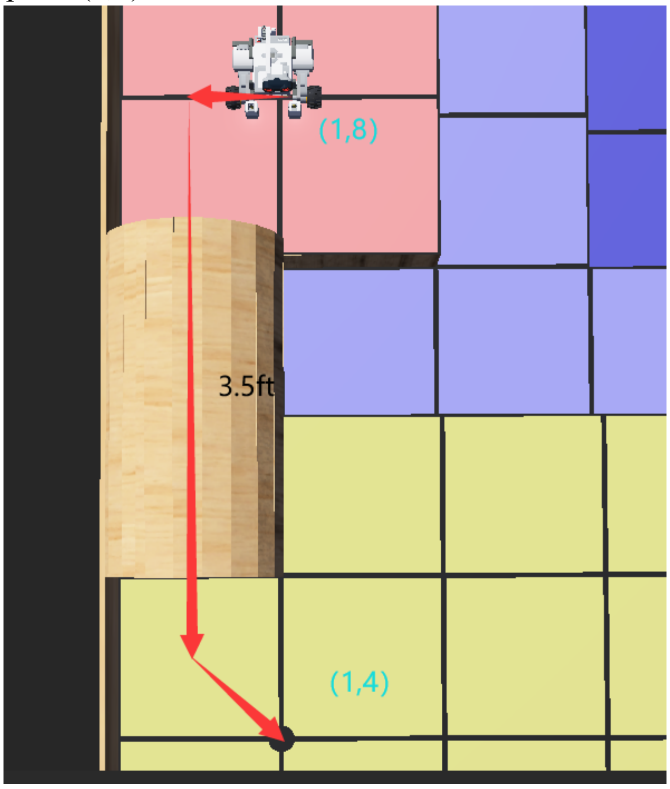


Figure 7.11: Software Version 1.2 Implementation

Software Design Version 1.3:

In Software Design Version 1.3, we added an algorithm for the robot to go over the overpass and go across the bottom of the overpass. We also add a PathManager class that generates the planned path according to the team color.
For going over the overpass, we let the robot move to the start point of the overpass, then move forward to climb the overpass. After that, the robot is trying to go downward. Then the robot will continuously move straight until the light sensors detect the overpass edge. The robot will move straight for 0.6ft, and then localize the nearest point (10,1). Then the robot will be navigated to destination (9,1).
For going across the bottom of the overpass, we just let the robot regard the overpass as an obstacle and let the robot find the tunnel and go across it.
PathManager will rearrange the waypoints according to the team color and find two special points to go over or go under overpass.

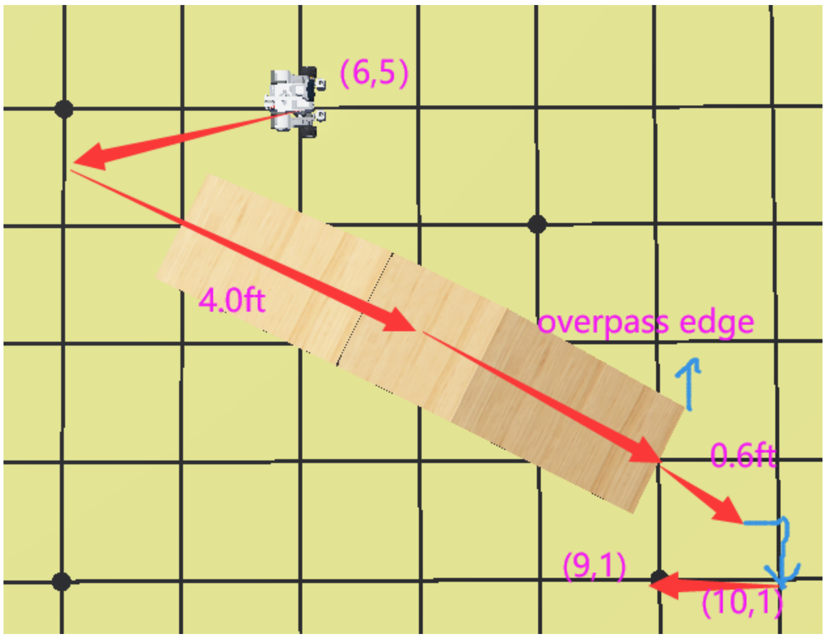


Figure 7.12: Software Version 1.3 Go over Overpass

The testing on Software Design V1.3 is shown in the table below:

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:
V1.3	Navigation	Pass	Test 1 of Navigation testing document
V1.3	Go Over Overpass	Conditional Pass	Test 1-5 of Go Over Overpass Testing Document
V1.3	Odometer	N/A	Test 1,2 of Odometer testing document
V1.3	Go under Overpass	Fail	Test 1 of Go under Overpass testing document

Software Design Version 1.4:

In Software Design Version 1.4, we revised the algorithm for going over the overpass and going under the overpass bottom. We also added a class bridge passer that helps to calculate the bridge position and let robots go through the bridge.
When performing the tests on Software Design Version 1.3, we found that it was dangerous for the robot to have a speed of 200 when it goes down the slope of the overpass. After performing tests on different speeds, we found 100 is the safe speed. So in Software Design Version 1.4, we set the speed of going down the slope of the overpass to 100, which makes the robot able to go down the slope safely.
In Software Design Version 1.4, we also update the algorithm for going across the overpass bottom. The robot will be navigated to the relative start point (7.5,4) of the bottom overpass, then it will travel to the relative endpoint (6.5,3) of the bottom overpass. Finally, it will travel to the destination waypoint.

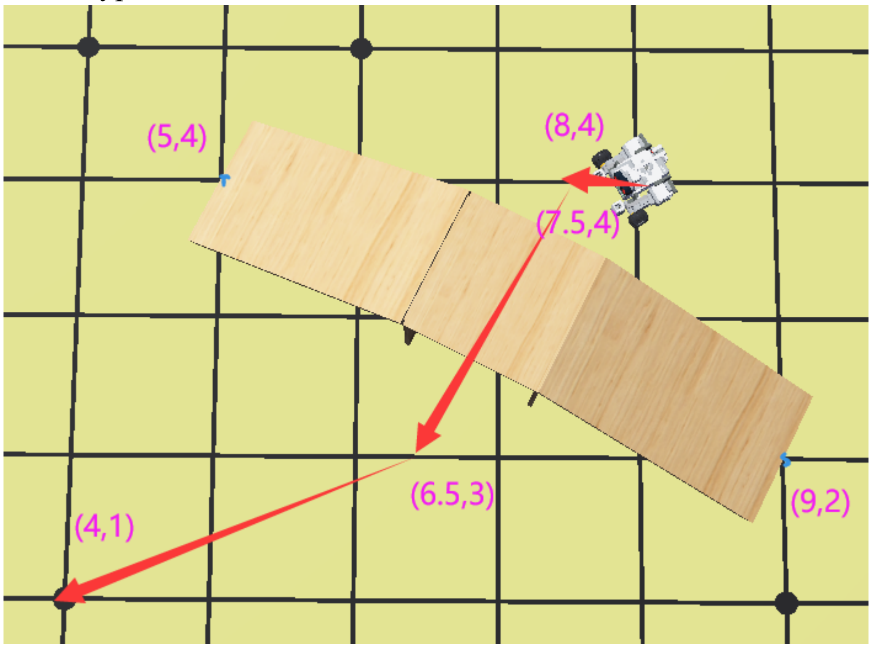


Figure 7.13: Software Version 1.4 Go under Overpass

The logic behind Bridge passer is that it will calculate distances between the robot position and bridge middle and judge if the point is on the edge on one of the land. If the point is on the edge, then we select it as a start point or endpoint.
The testing on Software Design V1.4 is shown in the table below:

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:
V1.4	Go Over Overpass	Pass	Test 6 of Go Over Overpass Testing Document
V1.4	Go under Overpass	Pass	Test 2,3 of Odometer testing document
V1.4	Bridge Passer	Partially Pass	Test 1 of Bridger Pass Testing Document
V1.4	Path Manager	Partially Pass	Test 1 of Path Manager Test Document

Software Design Version 1.5:
In software Version 1.5, we updated the algorithm for going over the overpass and going under the overpass. We also added new algorithms that enable the robot to return back to the start point.
For going above the overpass, we add several set Odometer value actions to the algorithm. We found that the odometer value will be influenced dramatically when the robot gets slightly stuck or the wheels slip during climbing the slope. Then the robot cannot distinguish its position and angle accurately when it ends going down the slope. The navigation system will crash so that the robot may fail to travel to the asserted destination. In software Version 1.5, we set the odometer at (10,1) because the localization must bring the robot to that point. Then the navigation system will navigate the robot to the destination waypoint according to the accurate odometer value.
For going under overpass, instead of using the hard code to let robots travel to a fixed start point and endpoint of overpass bottom, we use an algorithm to calculate a variational start point and endpoint of overpass bottom according to the given parameters (start point and endpoint of the overpass).
In software Version 1.5, we added an algorithm for returning back to the start point from the island. It is just let the robot be navigated to the start point calculated by the bridge passer, go across the bridge, and travel to the start point.
The testing on Software Design V1.5 is shown in the table below:

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:
V1.5	Return Back	Fail	Test 1 of Return Back testing document
V1.5	Pre Beta Demo Test	Fail	Test 1 of Pre-Beta Demo testing document
V1.5	Beta Demo Test	Fail	Test 1 of Beta Demo testing document

Software Design Version 1.6:
In the Software Design Version 1.6, we revise the bridge passer. During the tests on Version 1.5, the major software problem occurs when the robot returns back to the starting position. The robot passes the bridge along the blue path in Figure 7.16 and the green path in Figure 7.17. If the robot now has some special angles, the robot will hit the bridge edge and crash when it is trying to directly move to the entry of the bridge. In Version 1.6, As is shown in the red path in Figure 7.16 and the blue path in Figure 7.17, we set a pre-passing point and divide the path into several sub-paths which can reduce the possibility of crushing.
In the Software Design Version 1.6, we also revise the light localization. During the tests on Version 1.5, we found that the robot sometimes would fail to detect the black grid line so that it would continuously move forward and jump into the sea finally. So in Software Version 1.6, we decrease the robot's speed to 170 when it is trying to localize.

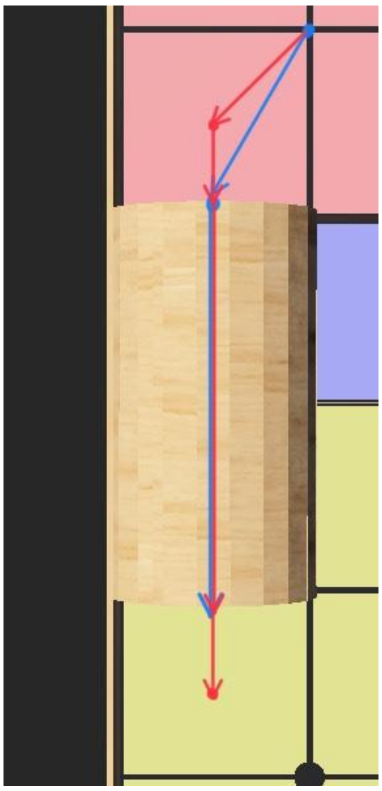


Figure 7.16: Software Version 1.6 Adjustment

The testing on Software Design V1.6 is shown in the table below:

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:
V1.6	Return Back	Test 2 Pass Test 3 Fail Test 4 Pass	Test 2,3,4 of Return Back testing document
V1.6	Pre Beta Demo Test	Fail	Test 2 of Pre-Beta Demo testing document
V1.6	Beta Demo Test	Fail	Test 2 of Beta Demo testing document

Software Design Version 1.7:

A problem about Software Design Version 1.6 was found during the tests of the Pre-Beta Demo Test and Beta Demo Test. During the test, after going crossing the bridge, the robot will localize to the (1,4) point. If the asserted waypoint is (1,4), the robot will not realize that it is at (1,4) right now. The robot then will travel to some other points and deviate from the planned path.

In the Software Design Version 1.7, we set a “Before-returning Point”. As shown in Figure 7.17, after arriving at the last waypoint, the robot will first navigate to the Before-returning Point with light localization, so the angle will be correct. Then the robot will follow the red path and return back.

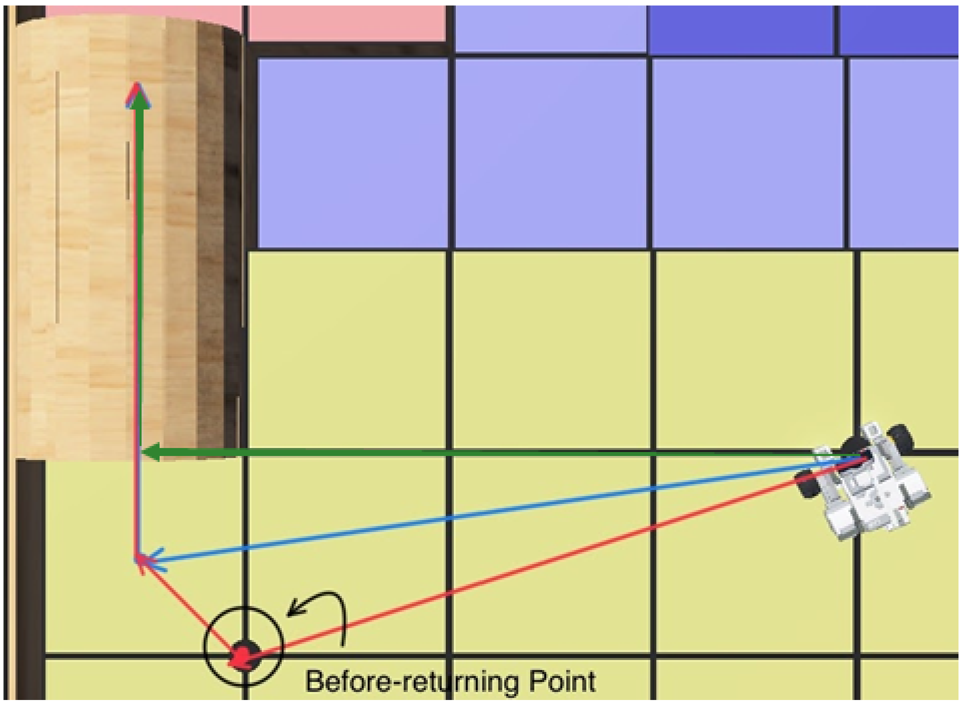


Figure 7.17: Software Version 1.7 Adjustment

In the Software Design Version 1.7, we added a judgment in the Navigation process. If the distance between the current point indicated by the odometer of the robot and the next waypoint is small enough (threshold is 0.05 ft), we consider that the robot has successfully reached the waypoint. We can start to travel to the next waypoint (if there exists) immediately.

The testing on Software Design V1.7 is shown in the table below:

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:
V1.7	Pre Beta Demo Test	Pass	Test 3 of Pre-Beta Demo testing document
V1.7	Beta Demo Test	Pass	Test 3 of Beta Demo testing document
V1.7	Obstacle Avoidance	Test 1 conditionally pass, Test 2 fail	Test 1,2 of Obstacle Avoidance testing document

<u>Software Design Version 1.8:</u>
In software Version 1.8, we updated the algorithm for going over the overpass and going under the overpass.
Firstly, we revise Path Manager to help to calculate the needed points to go over or under the overpass.
For going over the overpass, now we designed a flexible algorithm to handle this problem.
By extending the line between endpoint A and B of the overpass, we will get an intersect

point which is shown as "Pre-Overpass-Point" in Figure 7.18, in the grid where we go up the overpass. When we get this point, we can freely turn to the angle which the overpass points to and successfully go on the overpass.

For going under the overpass, now we designed a flexible algorithm to handle this problem. As shown in Figure 7.182, by calculating the midpoint and the slope of the overpass, we designed two points as pre-waypoints for going through the overpass.

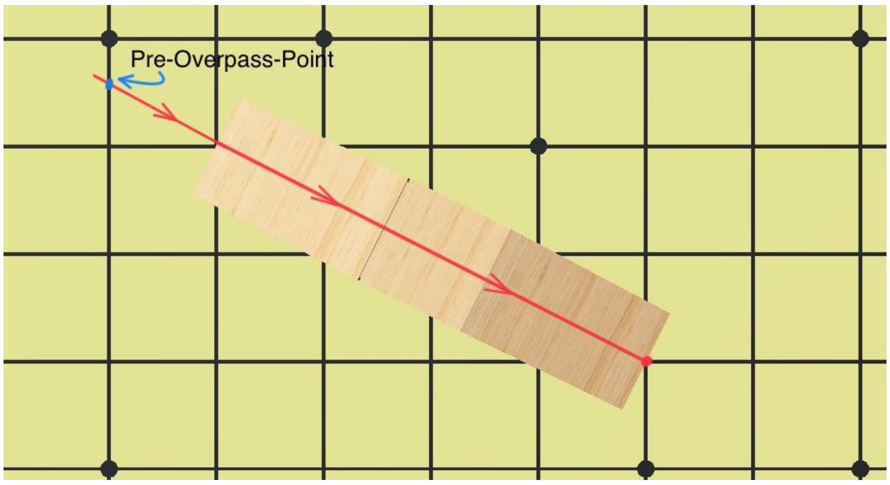


Figure 7.181: Software Version 1.8 Go Over Overpass

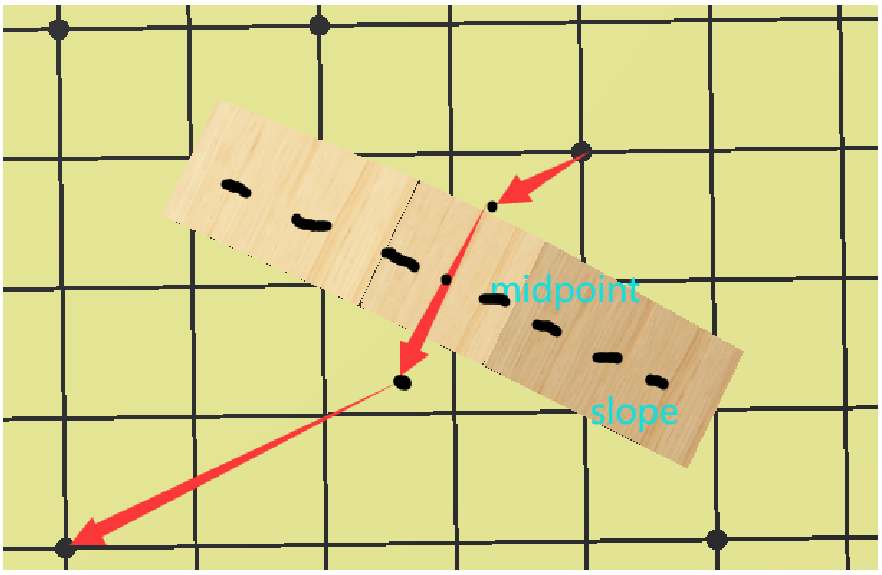


Figure 7.182: Software Version 1.8 Go UnderOverpass

The testing on Software Design V1.8 is shown in the table below:

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:
V1.8	Go Under OverPass	Test 4 Pass, Test 5 Fail	Test 4,5 of Go Under Overpass testing document

V1.8	Go Over Overpass	Conditional Pass	Test 8 of Go Over Overpass
V1.8	Path Manager	Pass	Test 2 of Path Manager

Software Design Version 1.9:
In software Version 1.9, we updated the algorithm for bridge passer. In our beta-demo, our robot fails because our bridge passer algorithm is not able to handle the case of horizontal tunnel. Now the bridge passer is able to handle cases of both horizontal and vertical tunnels. It can also recognize the team color now.
The testing on Software Design V1.9 is shown in the table below:

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:
V1.9	Bridge Passer	Conditional Pass	Bridger Pass Test document

Software Design Version 2.0:
In software Version 2.0, we updated the path manager class. During testing Software Design Version 1.8, in test 5 of the Go Under Overpass testing document, the path manager failed to tell the robot at which waypoint it should go overpass or underpass.
In V2.0, we update the algorithm for calculating overpass related waypoints. Currently, we check whether the line generated by the current and next waypoint will intersect with the line generated by the overpass endpoints. If there is an intersection, the pass manager will tell the robot that the robot should perform overpass related actions (i.e. go over or under the overpass) on this waypoint. If the angle difference between these two lines are between 45° and 135°, the robot should go underpass. Otherwise, it should go overpass.

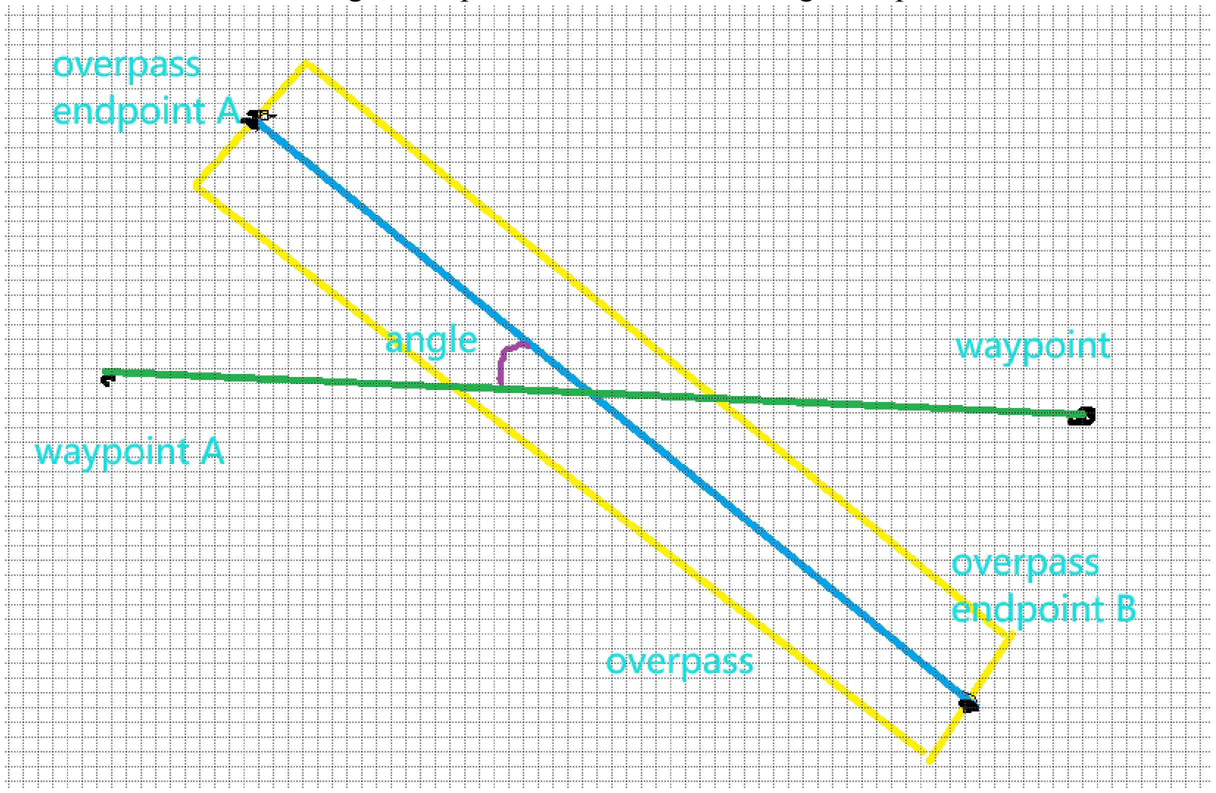


Figure 7.20: Software Version 2.0 Overpass Check

The testing on Software Design V2.0 is shown in the table below:

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:
V2.0	Path Manager (Overpass)	Pass	Test 3,5 of the Path Manager testing document
V2.0	Path Manager (Arrange Points)	Pass	Test 4 of the Path Manager testing document
V2.0	Go Under Overpass	Pass	Test 6,7 of the Go Under Overpass testing document
V2.0	Pre Final	Fail	Test 1 of the Pre Final testing document
V2.0	Complete Testing	Fail	Test 1,2 of the Complete System testing document

Software Design Version 2.1:

In software Version 2.1, we updated the algorithm for the obstacle avoidance.
From the tests on obstacle avoidance in the Obstacle Avoidance testing document, we found that the robot cannot handle the case that the obstacle is not exactly on the path, but the robot will hit the obstacle if the robot directly travels. So in V2.1, instead of only checking the obstacles in the front of the robot, the robot will check whether there are obstacles on the left or right. If there is an obstacle on the left, as is shown in Figure 7.21, the robot will design a path in the right of the robot and avoid it (purple path).

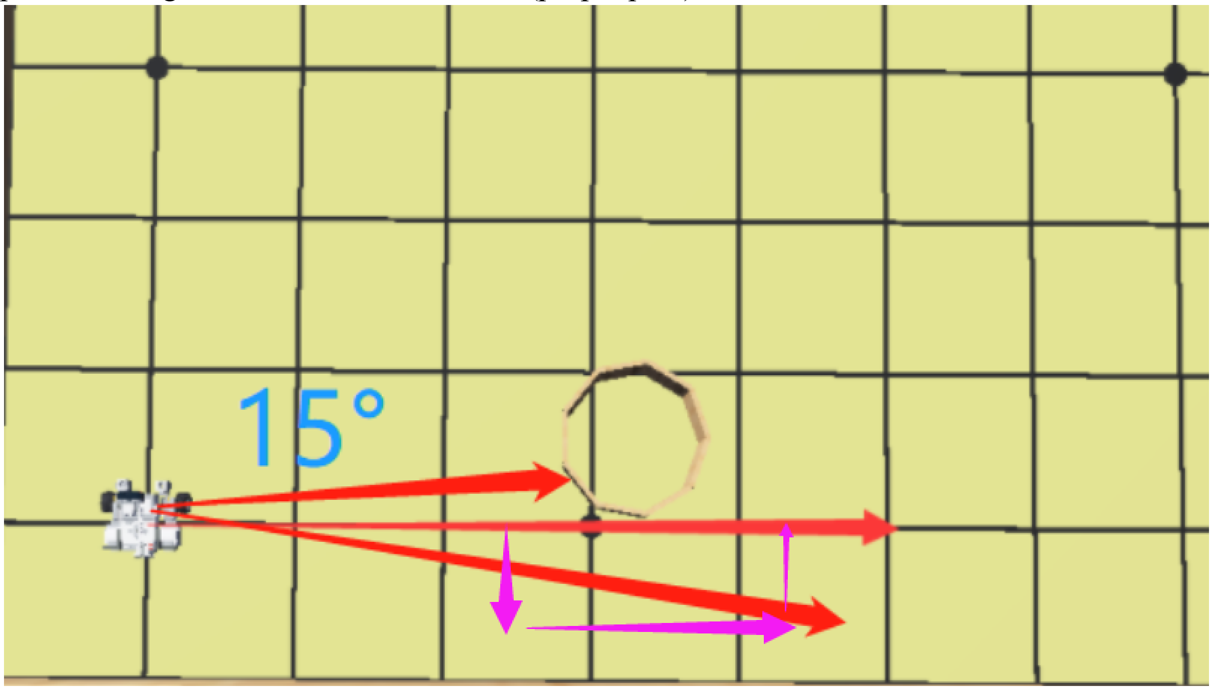


Figure 7.21: Software Version 2.1 Avoid Obstacle

The testing on Software Design V2.1 is shown in the table below:

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:

V2.1	Obstacle Avoidance	Test 3 Pass Test 4 Fail	Test 3,4 of Obstacle Avoidance testing document

Software Design Version 2.2:
In Software Version 2.2, we add a "After-Tunnel-Localization Point" near the end of the tunnel. Because after passing the tunnel, there are large deviations on the odometer from the actual position. So we need to use the light localization to correct the position and the robot will be localized to that point. As shown in Figure 7.22, the robot will pass the tunnel through the red path. Then the robot will localize to the point which the blue arrow points to.

In Software Version 1.7, "Before-returning Point" is similar to "After-Tunnel-Localization Point". However, in Software 1.7 we ignore this point and will vary according to the position of the tunnel. In the current version, the point can be calculated correctly.

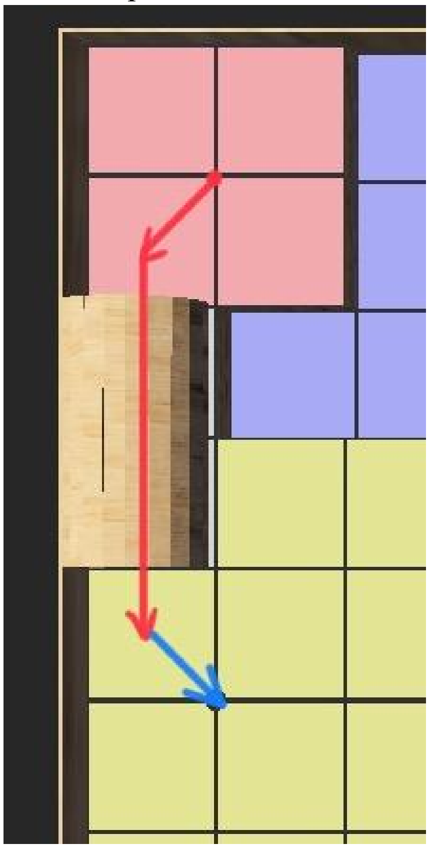


Figure 7.22: After-Tunnel-Localization Point

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:
V2.2	Bridge Passer	Pass	Test 3 of Bridger Pass Testing Document

Software Design Version 2.3:
In Software Version 2.3, we update the algorithms for avoiding obstacles.
During the testing on obstacle avoidance of Software Version 2.1, we found that the robot cannot detect the sea when it tries to avoid the obstacle. The ultrasonic sensor's feedback can only tell the robot whether there is an obstacle on the avoid path. So the robot sometimes will

fall into the sea when it is avoiding the obstacle. In V2.3, we divide the avoid path into several segments. For each segment, it has a start point and end point. We would check whether these start and end points on the avoid path are on the island. If they are outside of the main island, we regard them as unreached points. In this case, the robot will redesign a path to avoid the obstacle.

Furthermore, we revise the methods used to localize after passing the tunnel. After it passes the tunnel, it will move toward “After-Tunnel-Localization Point” for a short distance. Then it will start localization. As in some cases, the tail of the robot might hit the edge of the tunnel while rotating, this change can keep the robot from a distance from the tunnel, hence avoiding crushing. Considering this will be the last version of software before our competition, we perform an integration test which tests the robot from the start position to “After-Tunnel-Localization Point”.

Software Design Version:	Name of Test:	Pass/Fail	Link to Testing Document:
V2.3	Complete System Test	Fail	Test 3,4 of Complete Testing Document
V2.3	Pre-Final Test	Pass	Test 2 of Pre-Final Test

8.0 DISCUSSION ON BETA DEMO

The beta demo revealed a fatal problem - our bridge passer failed to detect the correct points across the bridge. After analyzing our code, it shows that our distance calculating algorithm has some problems. So we decided to use another approach - we add a new parameter that determines if the point is on one side of the island. If the point is not adjacent to the side, then it cannot be selected as a start or endpoint.
The beta demo also told us that the demo world will be considerably different from the example world. Some passed functions may work on the example world but may fail during the demo. We need to avoid writing methods using hard code. We need to update some methods such as going above the overpass and going under the overpass to let it be able to function correctly regardless of the world map.