Software Requirements Specification

for

Robot Radar

Version 2.0 approved

Prepared by Aung Khant kyaw, Marc Maestri, Kaden Civiletto

CSC 380 at SUNY Oswego

12-1-2021

Table of Contents

Revision History

Name	**Date**	**Reason For Changes**	**Version**
Kaden, Marc, Aung	8/31	Start to sections 1 and 2	1.0
Kaden, Marc, Aung	9/9	Revisions on sections 1 and 2	1.1
Aung	9/16	Updated Diagrams on Analysis Models	1.2
Aung	9/23	Updated Use case & Sequence Diagram. Class Diagram	1.3
Kaden	10/1	Updated Diagram	1.4
Aung	10/8	Updated Diagrams and Behavior Diagram	1.5
Aung	10/15	Activity Diagram and UI mock up	1.6
Aung, Kaden	11/19	Quality Testing, Deployment Guide	1.7
Kaden	11/22	Section 4	1.8
Aung	11/30	Edits, Testing, Diagrams updates	1.9
Team	12/1	Final edits	2.0

1. Introduction

1.1 Purpose

This SRS document is designed to provide a detailed description of the Robot Radar system designed in the CSC 380 course at SUNY Oswego. This document will explain what the system will accomplish and how the system will accomplish this. This document will also explain the constraints of which the system is bound by, as well as how the system will react to those constraints. This document will also explain how the system is controlled and interfaced with.

1.2 Document Conventions

Main headers will be written in bolded 18 point Times New Roman font. Subheadings will be written in bolded 14 point Times New Roman font. Section bodies will be written in 11 point Times New Roman font.

1.3 Intended Audience and Reading Suggestions

This SRS document is intended for Doctor Bastian Tenbergan and whoever is interested in the software requirements, specifications, and development of the Robot Radar project. The rest of the Software Requirement Specification document is organized as follows:
Section 2: the overall description of the Radar System including deployment guide.
Section 3: the External Interface Requirements including user interface mock ups.
Section 4: the product functions in details.
Section 5: the test cases and results
Use Cases diagrams and UML Design Diagrams such as System Architecture and and Sequence Diagrams can be found in the Appendix.

1.4 Product Scope

The purpose of Robot Radar is to provide a map-like interface for the GoPiGo2 Robot to navigate through obstacles. The objectives of this project are to design and implement a 360 degree user interface that is 26 ft wide using the laser system on the Robot, GoPiGo2. The system shall allow the user to see obstacles around the robot. The system shall also "decide" if an obstacle is the same when seen by multiple robots. The system will not be a map that shows the dimensions of the physical space the robot is in. The system will not determine how to deal with obstacles on its own and will simply show the location of obstacles seen by the Robot.

1.5 References

https://docs.google.com/document/d/1VvWPzAACspI83DSgYqdKyj-DaIrwkJsNtMBvtr2fgkc/edit
https://www.dexterindustries.com/gopigo2-tutorials-documentation/
https://wiki.seeedstudio.com/Grove-Ultrasonic_Ranger/
https://github.com/DexterInd/GoPiGo

2. Overall Description

2.1 Product Perspective

The origin of the product is a new, self-contained product being developed by SUNY Oswego Computer Science Department as part of a Course Project. The project is to create a “radar” like map for the GoPiGo users. The product will allow the developers to learn and work with a GoPiGo robot, a Raspberry Pi, and a sensor system.

2.2 Product Functions

The Gopigo2’s motion will be controllable via a keyboard with WASD keys. The system will be capable of moving forward, and backwards, as well as turning left and right. The distance sensor is mounted on a pivot which is also capable of rotating left and right. The movement of the sensor will be controlled by the left arrow and right arrow key. The system will use the distance sensor to detect obstacles. The system will also generate a map of obstacles, which moves relative to the movement of the GoPiGo2, which will be presented to the user.

2.3 User Classes and Characteristics

General users: Users with experience using a keyboard or gamepad to control a computer system should be able to utilise the system.
Specific groups:

- ROV operators
- Rescue teams
- Firefighters
- Law enforcement

2.4 Operating Environment

The operating environment for the software development is GoPiGo2 and Raspberry Pi 2(Hardware). Software: operation system, python, browser, wireless server component (wifi, ethernet, bluetooth), any flat physical room that is 22 ft wide. Where will the application be like a wireless connection via a web with the robot.

2.5 Design and Implementation Constraints

- Because the Gopigo2 will be stored at Shineman, project developers will need to be physically on campus to test software on the system.
- The Raspberry pi 2 controller has 1GB of ram and a 900mhz quad-core CPU, meaning software will have to be efficient enough to run within these specifications.
- The system will be designed to work inside a building.
- The ultrasonic distance sensor has a maximum effective range of 350 cm, and a minimum effective range of 2cm. However, the accuracy of the sensor decreased after the range of 200 cm.
- Because the ultrasonic sensor is mounted on a servo, the scanner can pivot a maximum of 180 degrees, which allows the robot to scan a maximum area of 19.24 square meters, without repositioning the wheels.
- The ultrasonic sensor has a resolution of 1cm, which means that recording distance changes smaller than 1cm is not possible. The ultrasonic sensor can detect obstacles that are smooth, with a minimum surface area of 0.5 square meters.
- Because the device will be battery powered, it will have a maximum runtime depending on the type of battery used, and the amount of power drawn on the battery.
- The robot will have to operate on a flat surface, other terrains may cause the robot to have misreadings on data and/or limit the movement of the robot.
- Because the user interface will be accessible over wifi, the system will be limited by the range of the wireless hardware used.

2.6 User Documentation

Deployment Guide:

Step 1.) Travel to https://github.com/KadenCiviletto/GoPiGo2-Robot-Radar. Clone the repository to your GoPiGo2

Step 2.) Open your GoPiGo2's remote desktop and select 'Launch VNC'.

Step 3.) Open your terminal through VNC and travel to the correct location where you saved the GoPiGo2-Radar file by entering 'cd GoPiGo2-Radar'

Step 4.) Enter: 'python main.py' and the program should run.

Step 5.) Use the 'E' button for controls, measurements and more information regarding the use of the program. (Image of said controls is also listed below in Section 3.1, Figure 2).

3. External Interface Requirements

3.1 User Interfaces

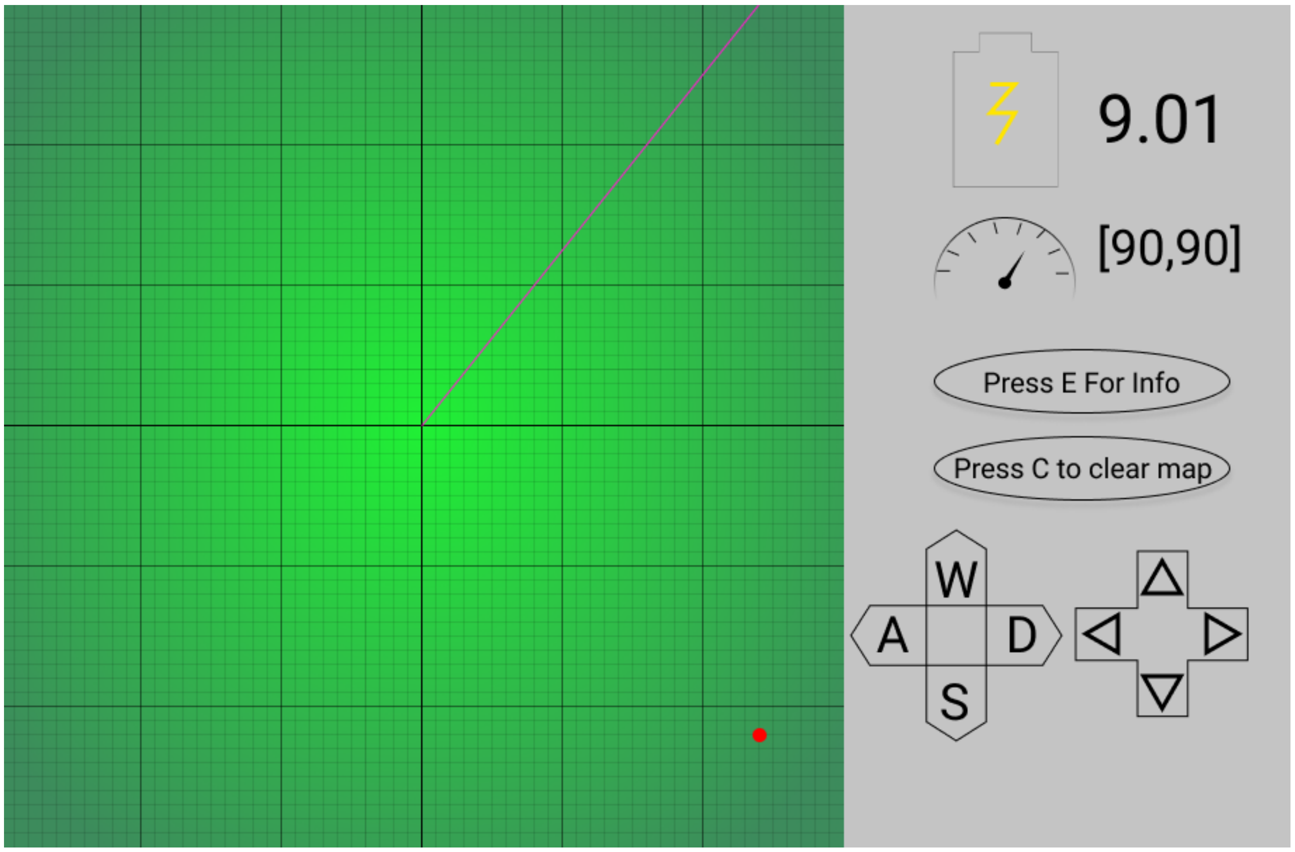


Figure 1: Main GUI with a mini map and controller

Figure 1 is the main user interface of the program. The map of the left is where the obstacles will be plotted. The pink line is the current angle that the robot is facing. The map is represented in a grid whereas each line tick is 10 cm. On the right side of the figure, the user can check the current voltage of the battery and the current speed of the robot.

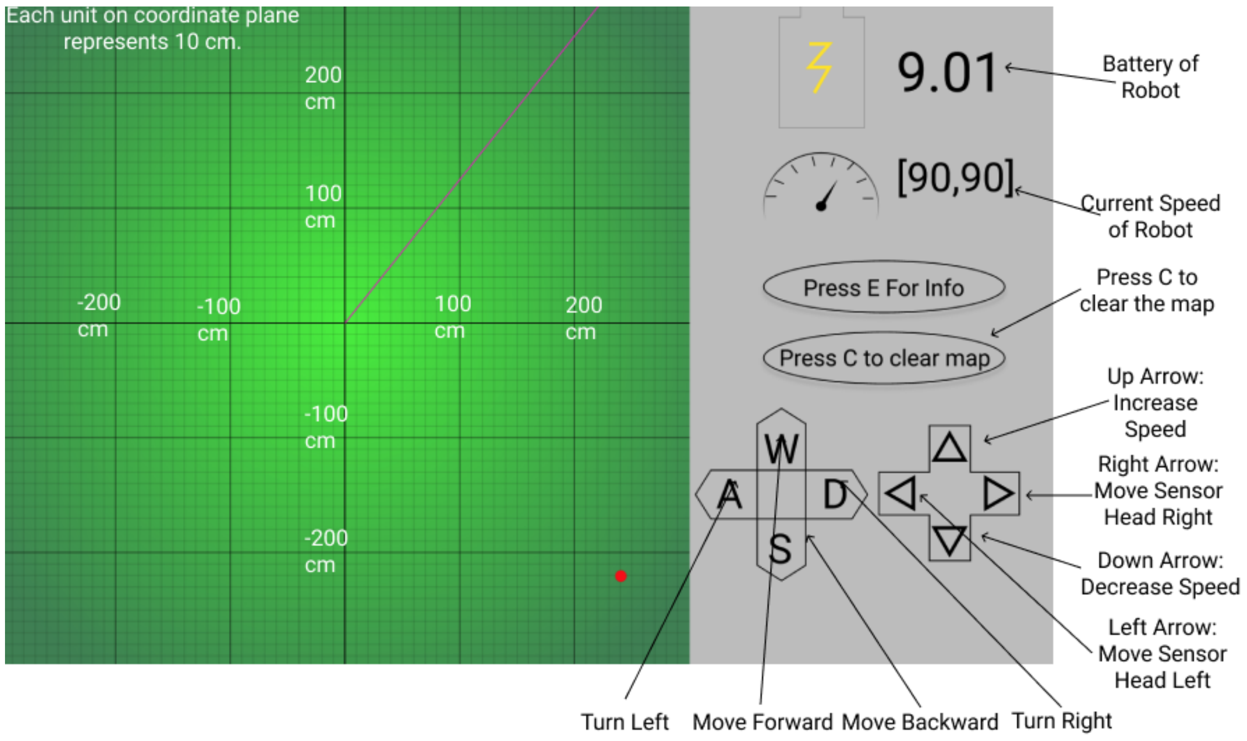


Figure 2: Information Page

Figure 2 is the information page which will be shown when the user pressed the "E" key on the keyboard for information about the system.

3.2 Hardware Interfaces

Through the gopigo python library, the system controls and uses external hardware on the Gopigo 2 such as servo, ultrasonic sensor, encoder, wheels, and battery. The system controls the servo on the Gopigo 2 to move the sensor. The system controls the wheels to move Gopigo 2. The system uses the ultrasonic sensor to sense obstacles in the Gopigo 2's path. The system uses the encoder to read and record the movement of Gopigo 2. The system uses the battery to check the voltage of Gopigo 2.

3.3 Software Interfaces

The system uses the Gopigo remote desktop (VNC) to open the program and run it. The system also uses external software libraries. gopigo python library found in https://github.com/DexterInd/GoPiGo/tree/master/Software/Python/gopigo.py is used to control the gopigo robot. Pygame (version 1.9.6) is used for the user interface and keyboard controls.

4. System Features

4.1 Display of GUI

4.1.1 HIGH PRIORITY. The GUI will be displayed to the user through the GoPiGo2's Virtual Desktop.

4.1.2 When the program is started the GUI will be displayed on the screen.

4.1.3 The program will be launched and the GUI will appear.

4.2 Display of Map

4.2.1 HIGH PRIORITY. The GUI will display an up to date map containing the obstacles that have been seen by the robot.

4.2.2 When the program is started the map will be displayed on the left of the GUI and it will be empty. The map will fill up as the user moves the sensor head and the obstacles will move relative to the robot movement forward, backwards, left or right.

4.2.3 The GUI will be launched and the map is displayed from the code. When the robot sensor head moves and scans, obstacles will be picked up by the system and signals will be sent to the Raspberry Pi inside of the GoPiGo2 robot containing points of obstacles that should be plotted and the system will plot said points and update the map accordingly.

4.3 Robot Movement

4.3.1 HIGH PRIORITY. The robot will move forward, backwards and turn left and right using a zero-point turn which allows the robot to perform a full 360 centered around itself.

4.3.2 When the user presses and/or holds the ‘W’, ‘A’, ‘S’, or ‘D’ key the robot will move. ‘W’ will cause the robot to move forward. ‘A’ will cause the robot to zero-point turn left. ‘S’ will cause the robot to move backward. ‘D’ will cause the robot to zero-point turn right.

4.3.3 The ‘W’, ‘A’, ‘S’, and ‘D’ keys will be pressed and a signal will be sent to the Raspberry Pi inside of the GoPiGo2 robot that will issue the command to move the robot in the respective way.

4.4 Sensor Movement

4.4.1 HIGH PRIORITY. The sensor head will move a full 180 degrees allowing the robot to look around while stationary.

4.4.2 When the user presses and/or holds the ‘left’ or ‘right’ arrow key the sensor head will move. Right moves the sensor head to the right. Left moves the sensor head to the left.

4.4.3 The ‘left’ and ‘right’ arrow keys will be pressed and a signal will be sent to the Raspberry Pi inside of the GoPiGo2 robot and it will issue the command to move the sensor head said way causing the sensor head to move 5 degrees per tick.

4.5 Speed Management

4.5.1 MEDIUM PRIORITY. The robot's speed can be controlled by the user through our GUI.

4.5.2 When the user presses and/or holds the ‘up’ or ‘down’ arrow key the speed will be adjusted. Up will increase the robot speed. Down will decrease the robot speed.

4.5.3 The ‘up’ and ‘down’ arrow keys will be pressed and a signal will be sent to the Raspberry Pi inside of the GoPiGo2 robot and it will issue the command to increase or decrease the speed by 10 units of speed per tick.

4.6 Sensor Scans

4.6.1 HIGH PRIORITY. The robot sensor will scan everytime the head is moved.

4.6.2 When the user moves the sensor head the robot will take 3 scans and averaging the 3 together to return us an accurate representation of where obstacles may be.

4.6.3 The robot will move and a signal will be received from the sensor that is sent to the Raspberry Pi inside of the GoPiGo2 robot and it will record this information and then issue a command to place it on our map.

4.7 Display Info UI

4.7.1 MEDIUM PRIORITY. The GUI will display information about all the different buttons, and commands and what they do.

4.7.2 When the user presses and/or holds the ‘E’ key an image will appear that shows all of the buttons and what they do.

4.7.3 The ‘E’ key will be pressed and a signal will be sent to the Raspberry Pi inside of the GoPiGo2 robot and it will issue the command to display the information screen.

4.8 Clear Map

4.8.1 MEDIUM PRIORITY. The map will be cleared.

4.8.2 When the user presses and/or holds the ‘C’ key all of the obstacles on the current map will be removed from the map.

4.8.3 The ‘C’ key will be pressed and a signal will be sent to the Raspberry Pi inside of the GoPiGo2 robot and it will issue the command to clear the map.

4.9 Servo Head Starts Facing Straight

4.9.1 MEDIUM PRIORITY. The robot sensor head will start facing straight relative to the robot.

4.9.2 When the program is launched the sensor head will automatically move to face 90 degrees which is the middle of the field of view.

4.9.3 When the program is launched the system inside the Raspberry Pi inside of the GoPiGo2 robot will issue the command to move the sensor head to location 90 degrees which is straight on the robot.

5. Other Quality Requirements

5.1 Performance Requirements

5.1.1 The refresh rate of the user interface shall be set to 60 frames per second. The refresh rate will be set 60 fps but when paired with raspberry pi and accuracy of the Ultrasonic Sensor on board, the smoothness of the map on UI shall be updated less than 3 sec.

5.2 Safety Requirements

5.2.1 Since the Ultrasonic Sensor is not mounted on the servo properly, the system shall limit the degree of angle that the servo rotates between 20 and 175 degrees. This will prevent the robot servo neck from breaking or harming the sensor.

5.3 Specification Based Testing

5.3.1 def obstacles_rotate(obstacles_list, angle_diff) in Main.py

Method:	obstacles_rotate		
Equivalence Classes			
Parameter	ID	Description	Representative
1. obstacles_list	1.1	A list of tuples where each tuple represents a obstacle point on the map	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]
	1.a	Not a list	abc

	1.b	null	null
2. angle_diff	2.1	a float	20.45
	2.2	an integer	0
	2.a	Not a float	(2,4)
	2.b	Not a integer	cat
	2.c	null	null

Boundary Value Analysis				
Parameter	ID	Boundary Values	Description	Test Case IDs
1. obstacles_list	BV 1.1.1	len(obstacles_list)=0	The size of the list is 0.	TC #8
	BV 1.1.2	len(obstacles_list)=1	The size of the list is 1.	TC #9
2. angle_diff	BV 2.1.1	float.max	The angle_diff is the maximum float number in python.	TC #10
	BV 2.1.2	float.max-1	The angle_diff is the maximum float number -1 in python.	TC #11
	BV 2.1.a	float.max+1	The angle_diff is the maximum float number + 1 in python.	TC #12
	BV 2.1.3	float.min	The angle_diff is the minimum float number in python.	TC #13
	BV 2.1.b	float.min-1	The angle_diff is the minimum float number -1 in python.	TC #14
	BV 2.1.4	float.min+1	The angle_diff is the minimum float number + 1in python.	TC #15

** In Python 3, the integer are unbounded.

Test Cases			
ID	obstacles_list	angle_diff	Expected Result
TC #1	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]	0	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]
TC #2	abc	0	Fail
TC #3	null	0	Fail
TC #4	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]	20.45	[(52.40847709926377, 140.54661692170305, 69.55), (-62.28428547452374, 136.35493310814988, 114.55), (-236.0111341475748, -83.80778339972638, 199.55), (179.34612271492395, 148.10458557088447, 39.55), (54.42194893537757, -323.4783632239955, 279.55)]
TC #5	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]	(2,4)	Fail
TC #6	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]	cat	Fail
TC #7	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]	null	Fail

TC #8	[]	0	[]
TC #9	[(0,150,90)]	0	[(0,150,90)]
TC #10	[(0,150,90)]	float.max	[(55.691912504970205, 139.27817805219433, -1.7976931348623157e+308)]
TC #11	[(0,150,90)]	float.max-1	[(55.691912504970205, 139.27817805219433, -1.7976931348623157e+308)]
TC #12	[(0,150,90)]	float.max+1	[(55.691912504970205, 139.27817805219433, -1.7976931348623157e+308)]
TC #13	[(0,150,90)]	float.min	[(9.184850993605149e-15, 150.0, 90.0)]
TC #14	[(0,150,90)]	float.min-1	[(9.184850993605149e-15, 150.0, 90.0)]
TC #15	[(0,150,90)]	float.min+1	[(9.184850993605149e-15, 150.0, 90.0)]

5.3.2 def gui_update(screen, robot_angle, obstacles) in gui.py

Method:	gui_update		
Equivalence Classes			
Parameter	ID	Description	Representative
1. screen	1.1	screen is pygame.display.	pygame.display
	1.a	Not a pygame.display.	20
	1.b	null	null
2. robot_angle	2.1	a float	20.45
	2.2	an integer	0
	2.a	Not a float	(2,4)
	2.b	Not a integer	cat
	2.c	null	null
3. obstacles	3.1	A list of tuples where each tuple represents a obstacle point on the map	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]
	3.a	Not a list	abc
	3.b	null	null

Boundary Value Analysis				
Parameter	ID	Boundary Values	Description	Test Case IDs
2. robot_angle	BV 2.1.1	float.max	The angle_diff is the maximum float number in python.	TC #11
	BV 2.1.2	float.max-1	The angle_diff is the maximum float number -1 in python.	TC #12

	BV 2.1.a	float.max+1	The angle_diff is the maximum float number + 1 in python.	TC #13
	BV 2.1.3	float.min	The angle_diff is the minimum float number in python.	TC #14
	BV 2.1.b	float.min-1	The angle_diff is the minimum float number -1 in python.	TC #15
	BV 2.1.4	float.min+1	The angle_diff is the minimum float number + 1in python.	TC #16
3. obstacles	BV 3.1.1	len(obstacles_list)=0	The size of the list is 0.	TC #9
	BV 3.1.2	len(obstacles_list)=1	The size of the list is 1.	TC #10

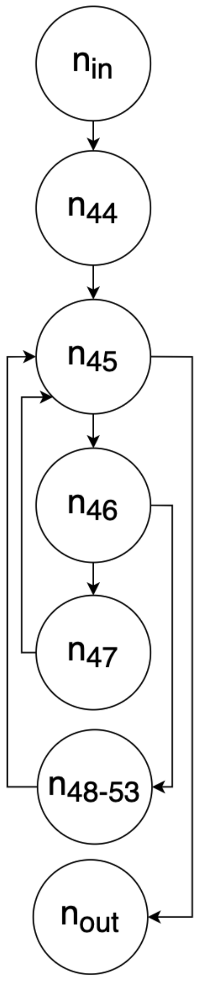
** pygame.display does not have a boundary

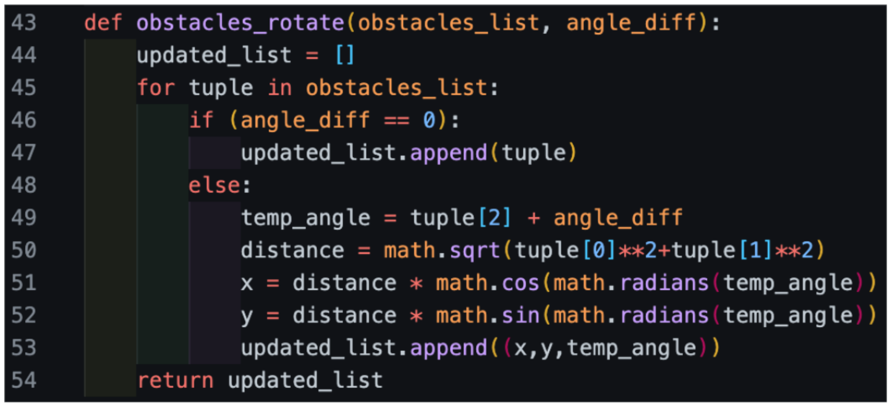
Test Cases				
ID	screen	obstacles	robot_angle	Expected Result
TC #1	pygame.display	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]	20.45	GUI updated
TC #2	20	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]	45	Fail
TC #3	null	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]	45	Fail
TC #4	pygame.display	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]	0	GUI updated
TC #5	pygame.display	[(0,150,90), (-106,106,135), (-15,-250,220), (100,210,60), (200,-260,300)]	(2,4)	Fail
TC	pygame.display	[(0,150,90), (-106,106,135),	cat	Fail

#6		(-15,-250,220), (100,210,60), (200,-260,300)]		
TC #7	pygame.display	null	20.45	Fail
TC #8	pygame.display	abc	20.45	Fail
TC #9	pygame.display	[]	0	GUI updated
TC #10	pygame.display	[(0,150,90)]	0	GUI updated
TC #11	pygame.display	[(0,150,90)]	float.max	GUI updated
TC #12	pygame.display	[(0,150,90)]	float.max-1	GUI updated
TC #13	pygame.display	[(0,150,90)]	float.max+1	GUI updated
TC #14	pygame.display	[(0,150,90)]	float.min	GUI updated
TC #15	pygame.display	[(0,150,90)]	float.min-1	GUI updated
TC #16	pygame.display	[(0,150,90)]	float.min+1	GUI updated

5.4 Control Flow Based Testing

5.4.1 obstacles_rotate(obstacles_list, angle_list)





```
43  def obstacles_rotate(obstacles_list, angle_diff):
44      updated_list = []
45      for tuple in obstacles_list:
46          if (angle_diff == 0):
47              updated_list.append(tuple)
48          else:
49              temp_angle = tuple[2] + angle_diff
50              distance = math.sqrt(tuple[0]**2+tuple[1]**2)
51              x = distance * math.cos(math.radians(temp_angle))
52              y = distance * math.sin(math.radians(temp_angle))
53              updated_list.append((x,y,temp_angle))
54      return updated_list
```

5.4.1: a) Statement Testing (C_0-Test)

Test Case ID	Parameters		Coverage	
	Obstacles_list	angle_diff	path	%
TC#1	[]	2	n_{in} - n_{44} - n_{45} - n_{out}	4/7≈ 57%
TC#2	[(0,150,90), (-106,106,135)]	0	n_{in} - n_{44} - n_{45} - n_{46} - n_{47} - n_{45} - n_{46} - n_{47} - n_{45} - n_{out}	6/7≈ 88%%
TC#3	[(0,150,90), (-106,106,135)]	20	n_{in} - n_{44} - n_{45} - n_{46} - n_{48-53} - n_{45} - n_{46} - n_{48-53} - n_{45} - n_{out}	7/7=100%

Add more rows, if necessary.

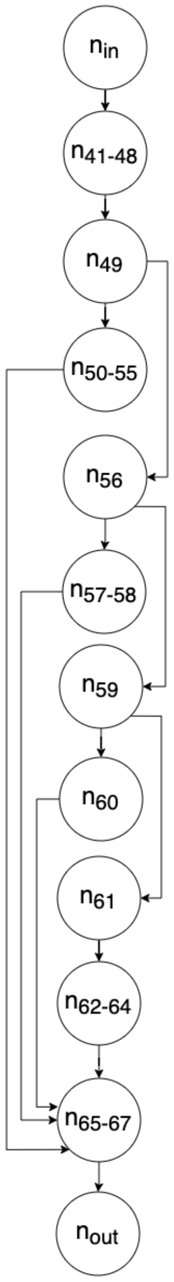
5.4.1: b) Branch Testing (C_1-Test)

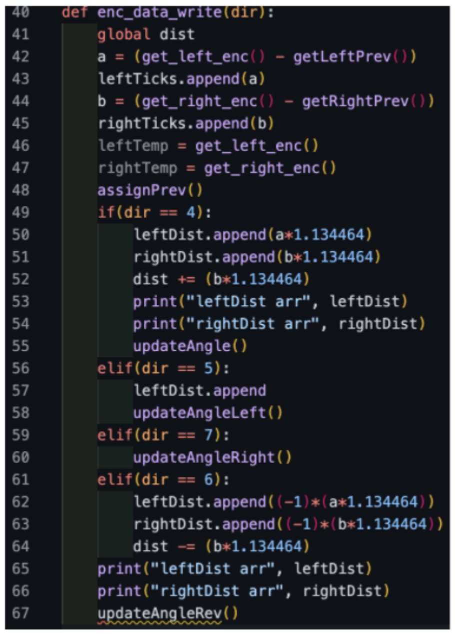
N_{45}: 2 Branches
N_{46}: 2 Branches

Test Case ID	Parameters	Coverage

	Obstacles_list	angle_diff	path	%
TC#4	[]	2	n_{in} - n_{44} - n_{45} - n_{out}	¼=25%
TC#5	[(0,150,90), (-106,106,135)]	0	n_{in} - n_{44} - n_{45} - n_{46} - n_{47} - n_{45} - n_{46} - n_{47} - n_{45} - n_{out}	¾=75%
TC#6	[(0,150,90), (-106,106,135)]	20	n_{in} - n_{44} - n_{45} - n_{46} - $n_{48\text{-}53}$ - n_{45} - n_{46} - $n_{48\text{-}53}$ - n_{45} - n_{out}	4/4 = 100%

5.4.2 enc_data_write(dir)





```
40  def enc_data_write(dir):
41      global dist
42      a = (get_left_enc() - getLeftPrev())
43      leftTicks.append(a)
44      b = (get_right_enc() - getRightPrev())
45      rightTicks.append(b)
46      leftTemp = get_left_enc()
47      rightTemp = get_right_enc()
48      assignPrev()
49      if(dir == 4):
50          leftDist.append(a*1.134464)
51          rightDist.append(b*1.134464)
52          dist += (b*1.134464)
53          print("leftDist arr", leftDist)
54          print("rightDist arr", rightDist)
55          updateAngle()
56      elif(dir == 5):
57          leftDist.append
58          updateAngleLeft()
59      elif(dir == 7):
60          updateAngleRight()
61      elif(dir == 6):
62          leftDist.append((-1)*(a*1.134464))
63          rightDist.append((-1)*(b*1.134464))
64          dist -= (b*1.134464)
65      print("leftDist arr", leftDist)
66      print("rightDist arr", rightDist)
67      updateAngleRev()
```

5.4.2: a) Statement Testing (C_0-Test)

Test Case ID	Parameters	Coverage	
	dir	path	%
TC#1	4	n_{in} - $n_{41\text{-}48}$ - n_{49} - $n_{50\text{-}55}$ - $n_{65\text{-}67}$- n_{out}	6/12 = 50%
TC#2	5	n_{in} - $n_{41\text{-}48}$ - n_{49} - n_{56} - $n_{57\text{-}58}$ - $n_{65\text{-}67}$- n_{out}	8/12 = 66%
TC#3	6	n_{in} - $n_{41\text{-}48}$ - n_{49} - n_{56} - n_{59} - n_{60} - $n_{65\text{-}67}$- n_{out}	10-12 = 83%
TC#4	7	n_{in} - $n_{41\text{-}48}$ - n_{49} - n_{56} - n_{59} - n_{61} - $n_{62\text{-}64}$ - $n_{65\text{-}67}$ - n_{out}	12/12 = 100%

Add more rows, if necessary.

5.4.2: b) Branch Testing (C_1-Test)

N_{49}: 2 Branches
N_{56}: 2 Branches
N_{59}: 2 Branches
N_{61}: 2 Branches

Test Case ID	Parameters	Coverage	
	dir	path	%
TC#5	4	n_{in} - $n_{41\text{-}48}$ - n_{49} - $n_{50\text{-}55}$ - $n_{65\text{-}67}$- n_{out}	⅛ = 12%
TC#6	5	n_{in} - $n_{41\text{-}48}$ - n_{49} - n_{56} - $n_{57\text{-}58}$ - $n_{65\text{-}67}$- n_{out}	⅜ =37%
TC#7	**6**	n_{in} - $n_{41\text{-}48}$ - n_{49} - n_{56} - n_{59} - n_{60} - $n_{65\text{-}67}$- n_{out}	⅝ =62%
TC#8	7	n_{in} - $n_{41\text{-}48}$ - n_{49} - n_{56} - n_{59} - n_{61} - $n_{62\text{-}64}$ - $n_{65\text{-}67}$ - n_{out}	8/8 = 100%

Appendix: Analysis Models

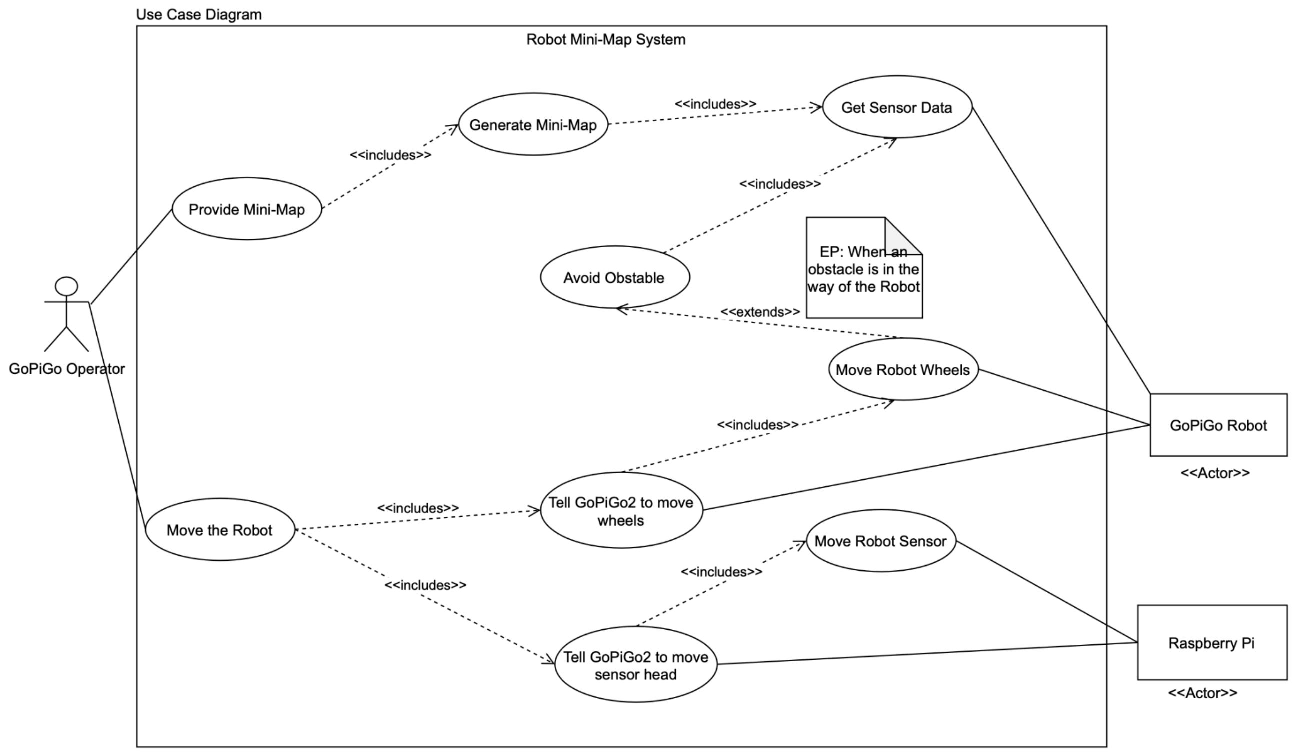


Figure 3: Use Case Diagram

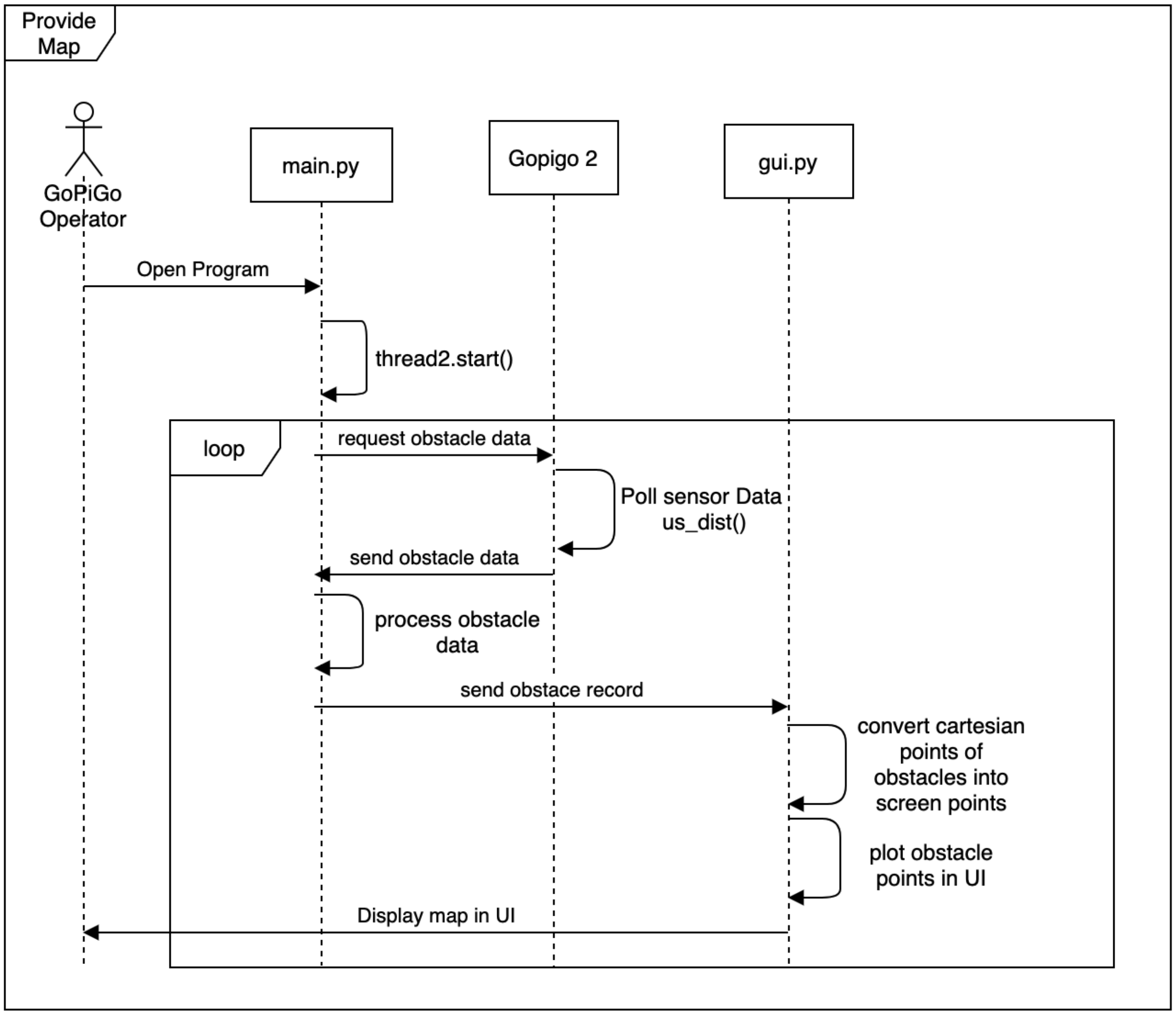


Figure 3: Sequence Diagram of Provide Map case

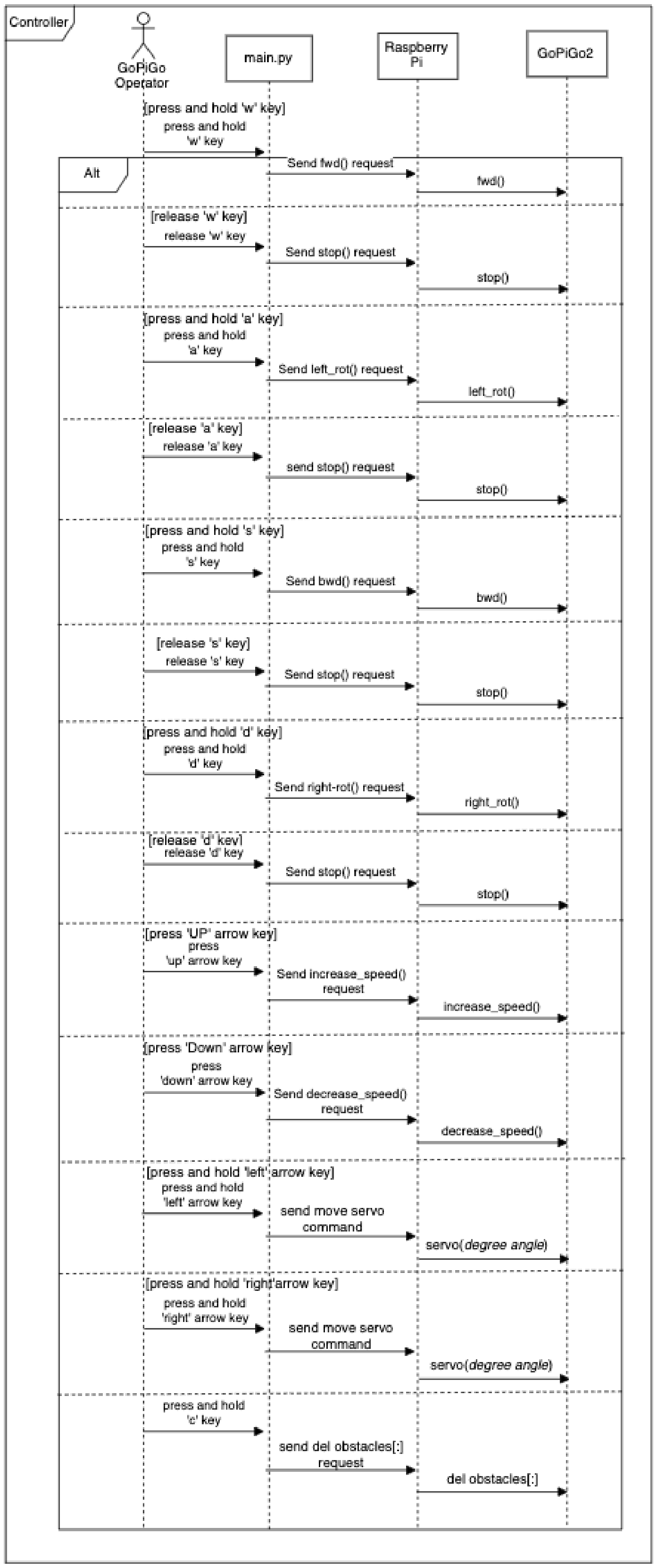


Figure 4: Sequence Diagram of Controller

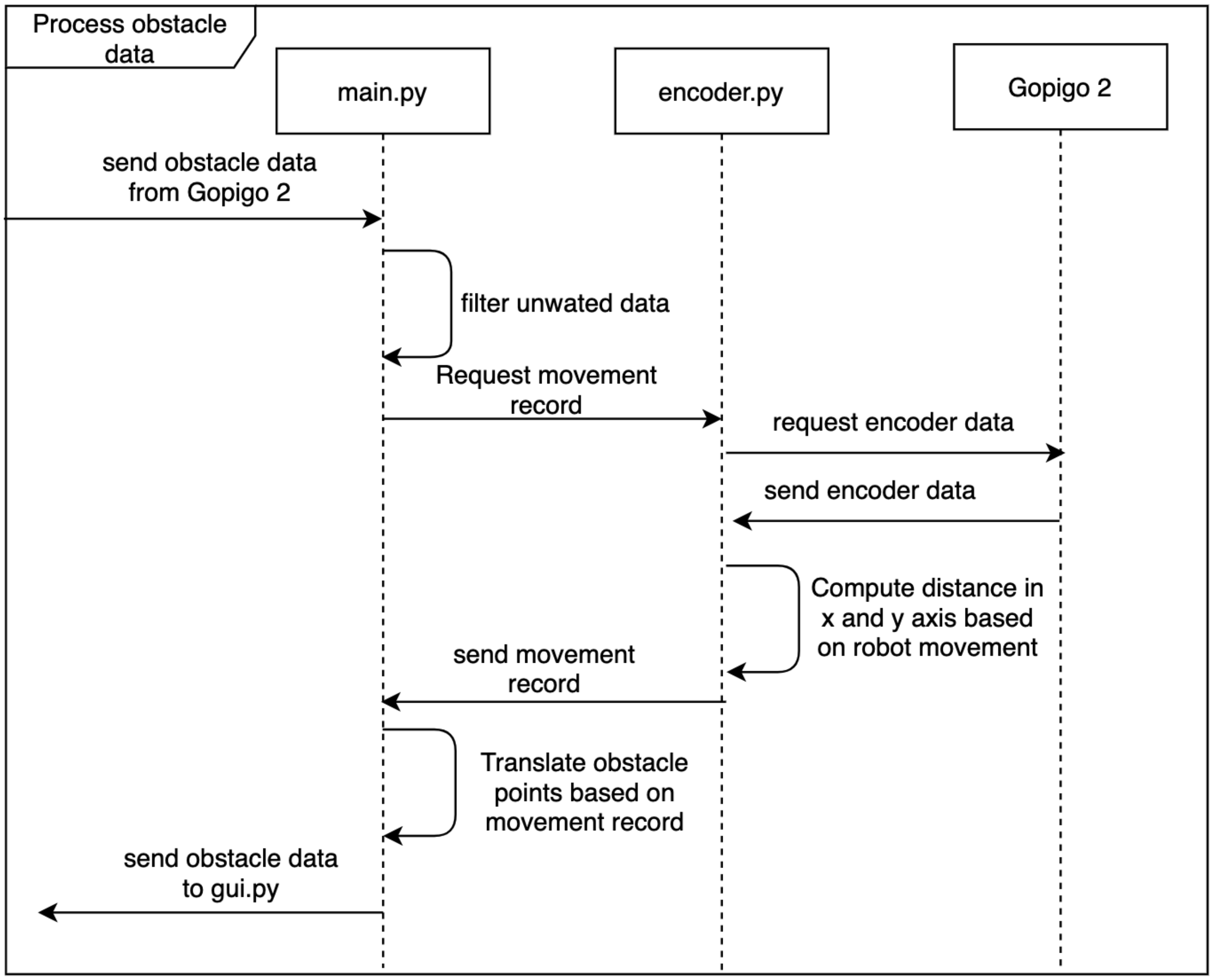

Process obstacle data
main.py
encoder.py
Gopigo 2
send obstacle data from Gopigo 2
filter unwated data
Request movement record
request encoder data
send encoder data
Compute distance in x and y axis based on robot movement
send movement record
Translate obstacle points based on movement record
send obstacle data to gui.py

Figure 5: Sequence diagram of Process Obstacle data

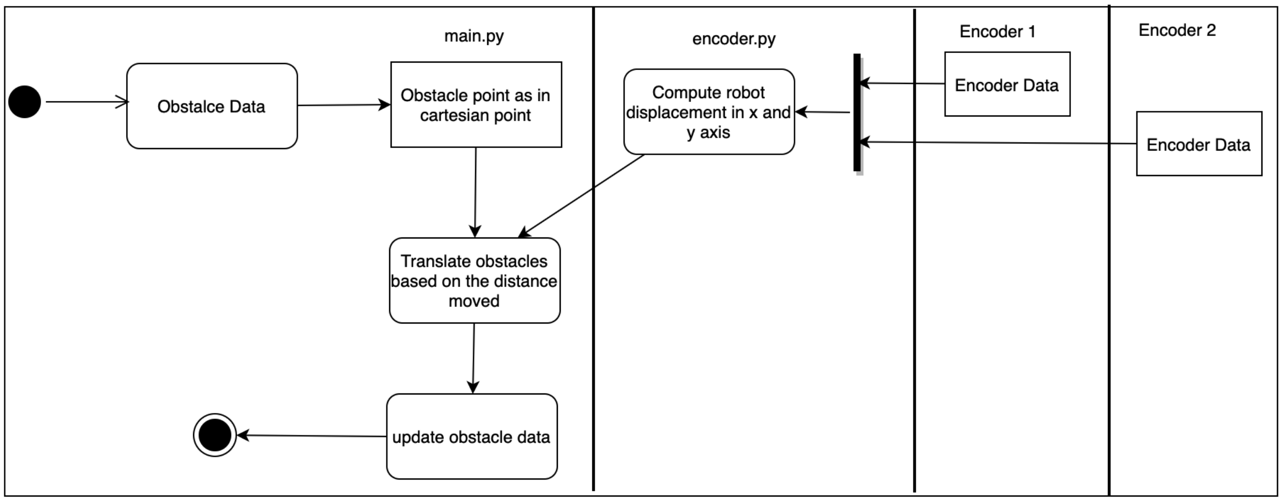

main.py
encoder.py
Encoder 1
Encoder 2
Obstalce Data
Obstacle point as in cartesian point
Compute robot displacement in x and y axis
Encoder Data
Encoder Data
Translate obstacles based on the distance moved
update obstacle data

Figure 6: Activity diagram 1

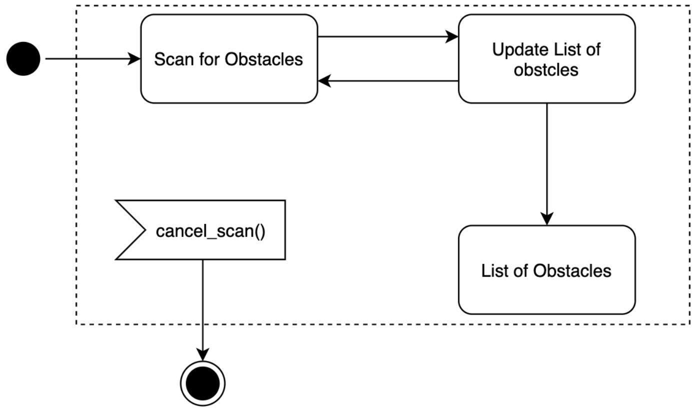


Figure 7: Activity diagram 2

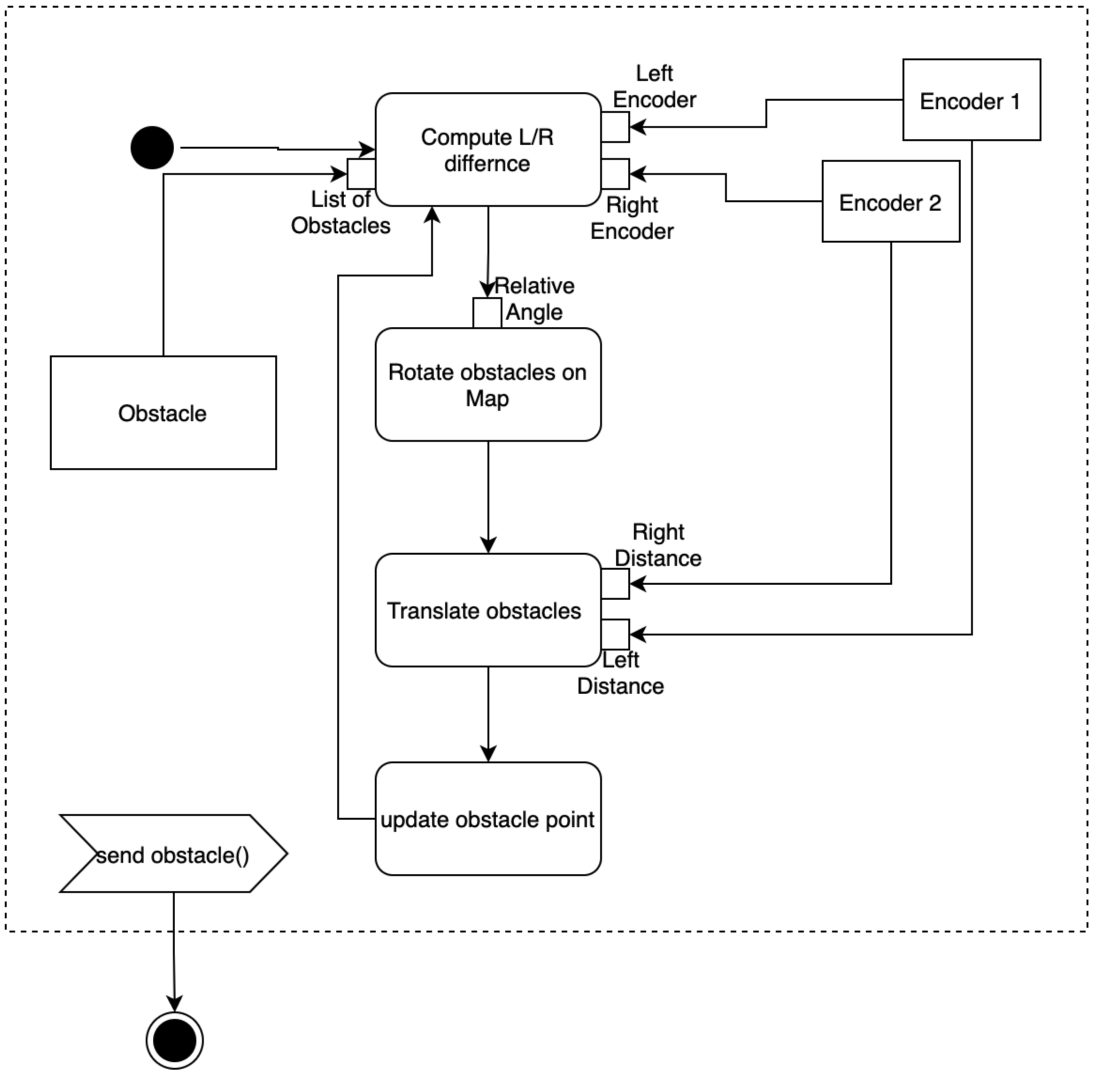


Figure 8: Activity diagram 3

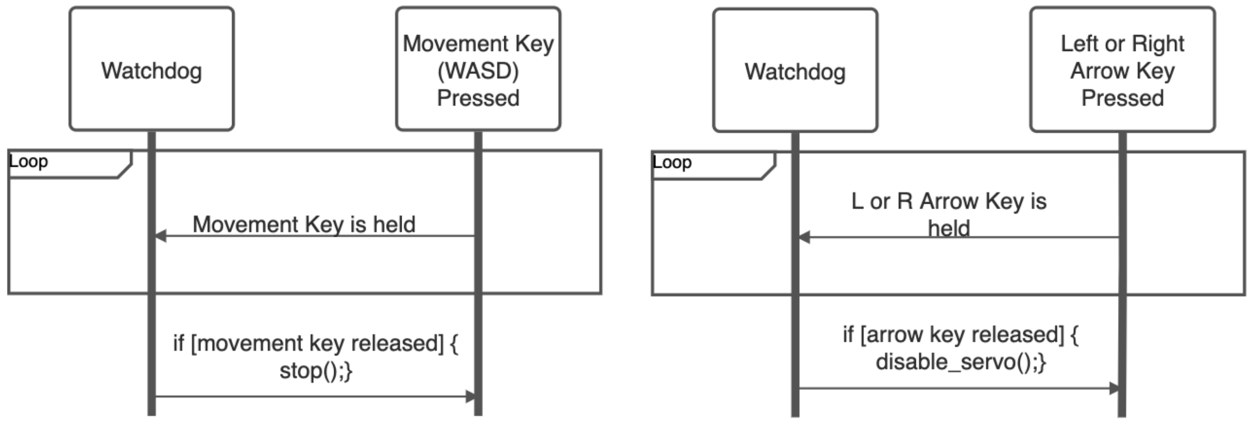

Watchdog
Movement Key (WASD) Pressed
Loop
Movement Key is held
if [movement key released] { stop();}
Watchdog
Left or Right Arrow Key Pressed
Loop
L or R Arrow Key is held
if [arrow key released] { disable_servo();}

Figure 9: Watchdog pattern diagram

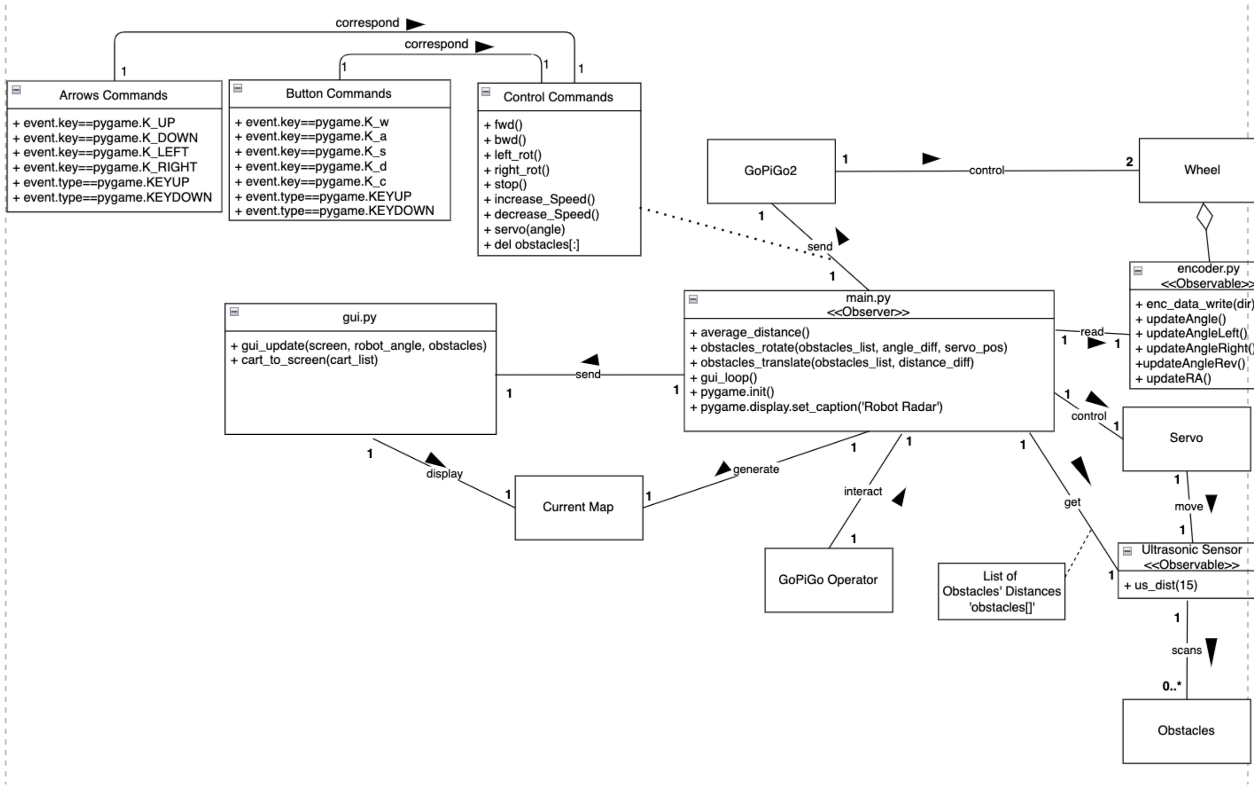


Figure 10: Class diagram / System Architecture