CSC 385 - Software Quality.

Robot Radar (From CSC 380)

Quality Management Plan

Prepared By: Umang Patel and Sven Kappeler
Date created: September 25, 2022
Version No: v3.0

Table Of Contents

Document Change Control

The following is the document control for revisions to this document.

Name	Date	Reason For Changes	Version
Umang	09/25/2022	initial edit to QMP	1.0
Sven	09/25/2022	initial edit to QMP	1.0
Umang	11/04/2022	added CFG DF	2.0
Sven	11/045/2022	added CFG DF	2.0
Umang	12/01/2022	Finals edit to QMP	3.0
Sven	12/015/2022	Finals edit to QMP	3.0

Definition

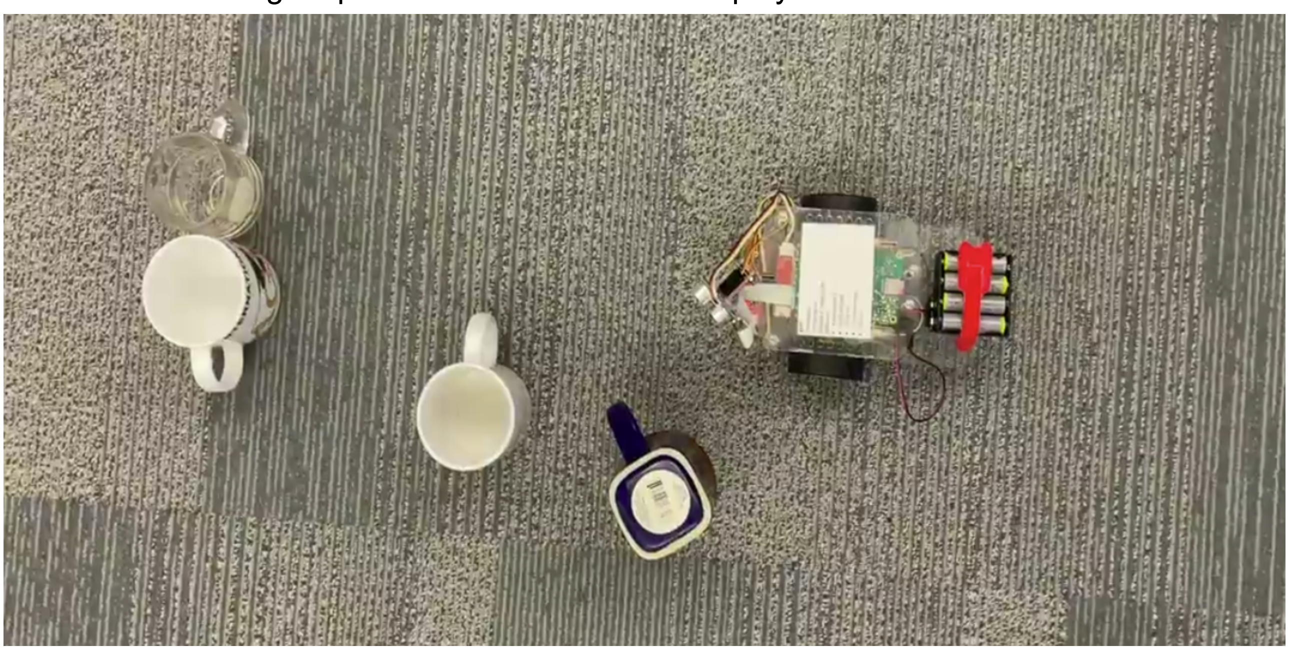
The following are definitions of terms, abbreviations, and acronyms used in this document.

Term	Definition
PMA	Project Management Advisor Web tool
GoPiGo	A Raspberry Pi-Controlled Robot with a servo and wheels
Servo	The mechanism that moves the ultrasonic sensor
UI	User Interface
SRS	System Requirements Specification
QMP	Quality Management Plan
HSV	Hue, Saturation and Value
SRS	software requirement specification document

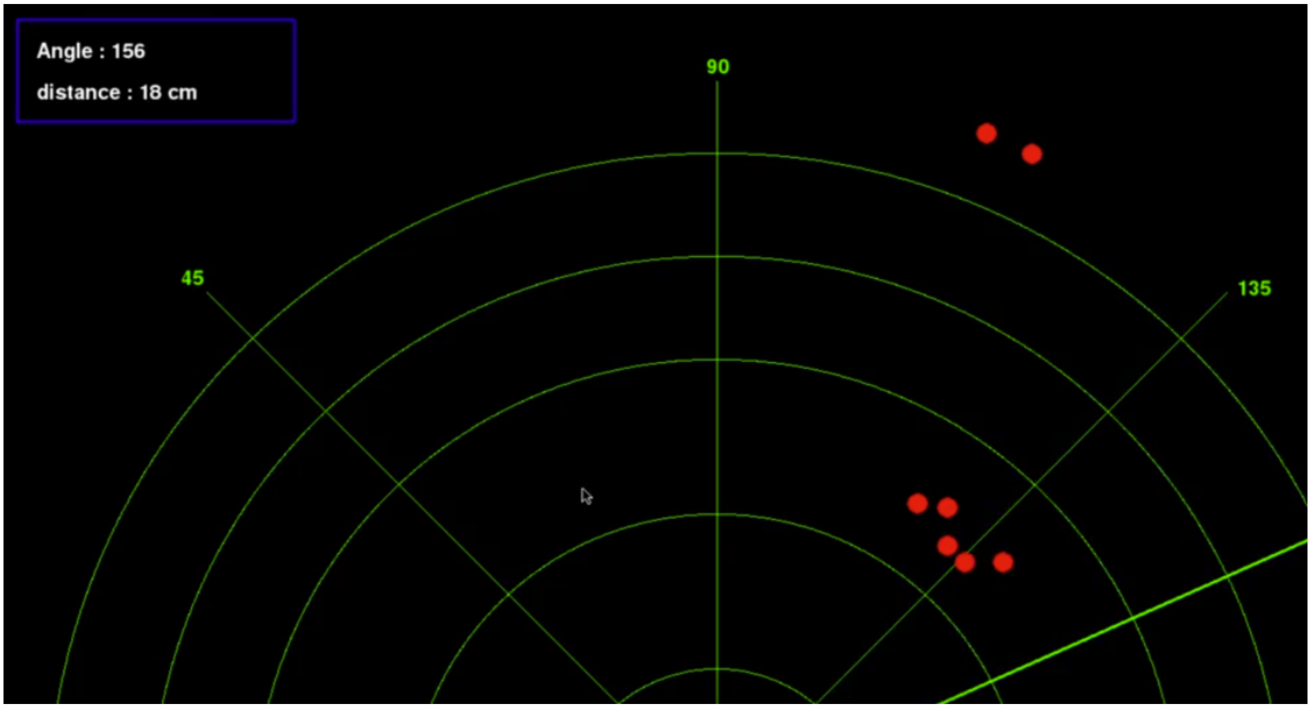
1. Introduction to Robot Radar

This is the project that we developed during our 380 in fall 2021. The project runs on the Raspberry Pi attached to the GoPiGo. The project was to develop a working radar system for the GoPiGo that scans objects and tracks them based on the robot's movements. whereby the robot will understand its environment and avoid collisions. We decided to use ultrasonic translateralization beacons to detect objects in the path and gain an understanding of its environment. The main reason to develop this was that the GoPiGO robot is a straightforward robot anyone can use but driving it carelessly can break it and be costly so we decided to make this program whereby the robot automatically applies brakes when an object is too close to the robot. We also decided to develop a user interface map so that the user can see that the robot does not drive into an object around its surroundings. So the objects would then be displayed on a user interface with colored dots on the radar that would be in its path. Red dot equal to too close and green dot being the farthest

Our implementation of this description was that we developed software that made a robot move in set increments, small enough that they seemed seamless to the user operating the robot. After each increment, the target's coordinates would get updated and move on the display.



(Robot mid-scan of surrounding objects)



(User Display including some target pings)

The most essential methods for our software are display.py and radar.py. The first is vital for setting up the user interface for our software, drawing the radar lines, and setting up the text boxes, and the latter is the one that is run to move the robot and to activate the scanning of the servo. Some secondary components are the color.py file used to determine the color of the targets and the target.py file, used in constructing the targets later on in the radar.py file.

Purpose of the system:

The Robot Radar program for GoPiGo robots allows the user of the robot to move the robot around thereby understanding its environment and avoiding collisions. The application uses ultrasonic translateralization beacons to detect objects in the path and gain an understanding of its environment. The object would then be displayed on a user interface with colored dots on the radar that would be in its path. It will stop automatically if an object is deemed to be too close, preventing a crash. The GoPiGo API is imported and used to interface with the robot, these methods are limited to sending movement commands to the robot and servo angle commands.

2. Quality Management Approach

The purpose of the Quality Management Plan (Plan) is to outline the following activities: define roles and responsibilities; provide reference documents and guidelines to perform the Quality Assurance (QA), and provide the standards, practices, and conventions used in carrying out QA, Quality Control (QC), and quality improvement activities for our CSC 380 Project; provide the tools, techniques, and methodologies to support QM activities and reporting.

The quality management plan identifies these key components of the project based on files that consist of important methods and their functions.

File	**Method**	**Function**
display.py	hsv2rgb(h, s, v)	Used to change color on depending the distance of object
	hsv3rgb(h,s,v)	Used to change color on depending the distance of object
	draw(radarDisplay, targets, angle, distance, fontRenderer)	Initializes the pygame's display window Draws the UI in the pygame's screen Loops through all target objects and draws them on the display Displays the servo's current angle
radar.py	us_map()	Moves the Servo to 0 Checks if objects are too close Waits for inputs to start moving
target.py	__init__(self, angle, distance)	Used for constructing Target Objects (~ Angle, Distance)
gopigo.py	servo(angle)	Used for relaying instructions to the servo to have the angle of the servo, and ultimately the ultrasonic sensor, move. its an API called used from gopigo

Referenced Delivered Software
"Robot Radar." *GitHub*, 28 Aug. 2021, github.com/asigdel29/RobotRadar.
https://github.com/asigdel29/RobotRadar "

3. Quality Management Objectives

The following are the quality objectives of the project that reflect the overall intentions to be applied to quality throughout the project.

- Identifies the activities, processes, and procedures used to manage quality.
- Defines the quality management methodologies, roles, and responsibilities required for the project
- Ensure project delivered conforms to SRS
- Defines the quality planning, Quality Assurance, Quality Control, and quality improvement processes, and procedures
- Project practices conform to recommended project management standards

4. Quality Team Roles & Responsibilities

The following identifies the quality-related responsibilities of the project team and lists specific quality responsibilities.

Name	Contact detail	Roles
Umang Patel	upatel2@oswego.edu	Quality Assurance
Sven Kappeler	skappele@oswego.edu	Quality Assurance

5. Project Quality Control

The focus of quality control is on the delivered project. Quality control monitors the project to verify that the delivered project is of acceptable quality and is complete and correct.

The following table identifies

- The major project aspects that will be tested satisfactory quality level.
- The quality standards and the correctness and completeness. Included are any organizational standards that need to be followed.
- The quality control activities will be executed to monitor the quality of the delivered project.
- How often or when the quality control activity will be performed

Quality Properties, Metrics, and Criteria

Quality Property	Definition	Metric	Criterion
Correctness	All test cases run have the same result as our expected results such as same (angle, color, distance, time, UI).	Evaluating Specification-Based Testing	The component is considered correct if: ● 100% of test cases with high criticality pass ● 90% of test cases pass with medium criticality pass ● 80% of test cases with low criticality pass
Efficiency	All test cases run without duplicate/unnecessary tasks being performed.	Performing Source Code-Based Testing and Data Flow Analysis on a Control Flow Diagram	The component is considered efficient if: ● 100% of test cases with high criticality pass ● 90% of test cases pass with medium criticality pass ● 80% of test cases with low criticality pass
Completeness	All methods and features described in the SRS document are present in the software	Evaluated against the relevant sections in the SRS documents, such as features criteria and expected behaviors	The component is considered complete if: ● No features or methods are missing (High Completeness) ● 10% of methods or features are missing (Medium Completeness) ● 20% of the outlined feature or methods are missing (Low Completeness)

6. Specification-Based Testing

The focus of quality assurance is on the processes used in the project. Quality assurance ensures that project processes are used effectively to produce quality project deliverables.

Specification-based Test

1. servo(angle)
2. init (Angle, distance)
3. hsv2rgb(h, s, v)
4. draw(radar display, targets, angle, distance, fontRenderer)
5. us_map()

6.1 servo(angle)

a) Equivalence Classes

Parameter: servo(angle)	**Equivalence Class ID**	**Description**	**Representative**
angle	1.1	180 ≥ angle ≥ 0	90
angle	1a	angle < 0	-20
angle	1b	angle > 180	190
angle	1.2	The case that angle is a float	33.33
angle	1c	The case that there is no input for angle	NULL
angle	1d	Angle is not a Float or and Int //Angle is a string.	“test”

b) Derived Test Cases

Test Case ID	(angle)	Exp. Result
TC#1	90	*Servo move to 90 degrees* (Straight ahead)
TC#2	-20	“IOError”
TC#3	185	“IOError”
TC#4	33.33	*Servo move to 30 degrees* (Leftish)
TC#5	NULL	“IOError”
TC#6	“test”	“IOError”
TF# 7	a	Error

c) Boundary Value Analysis:

Parameter	Boundary Values	Test Case ID(s)
angle	angle< 0	TC#2
angle	180< angle	TC#3
angle	MinInt-1,MinInt,MaxInt, MaxInt+1	TC#8-TC11

Additional Test Cases for Boundary Value Analysis

Test Case ID	(angle)	Exp. Result
TC#8	MinInt-1	Fail
TC#9	MinInt	Fail
TC#10	MaxInt	Fail
TC#11	MaxInt+1	Fail

6.2 init (angle, distance)

a) Equivalence Classes

Parameter: init(Angle, distance)	**Equivalence Class ID**	**Description**	**Representative**
angle	2.1	180 ≥ angle ≥ 0	90
angle	2a	angle < 0	-20
angle	2b	angle > 180	190
angle	2.2	The case that angle is a float	33.33
angle	2c	The case that there is no input for angle	NULL
angle	2d	Angle is not a Float or and Int //Angle is a string.	“test”
distance	3.1	1000 ≥ angle ≥ 0	100
distance	3a	distance is null	NULL

distance	3b distance is string	"java"
distance	3c *distance* < 0	-20
distance	3c *distance* > 1000	3000

b) Derived Test Cases

Test Case ID	**(angle)**	**(distance)**	**Exp. Result**
TC#12	90	100	Object is marked on 90 degrees at distance 100 is stored in array Target. (passed test case)
TC#13	null	100	error
TC#14	120	null	error
TC#15	java	100	error
TC#16	100	java	error
TC#17	-20	100	error
TC#18	80	3000	error
TC#19	90	-100	error
TC#20	260	300	error

c) Boundary Value Analysis:

Parameter	Boundary Values	Test Case ID(s)
angle	angle< 0	TC#17
angle	180< angle	TC#20
angle	MinInt-1,MinInt,MaxInt, MaxInt+1	TC#21-TC24
distance	distance< 0	TC#19
distance	1000< distance	TC#18
angle	MinInt-1,MinInt,MaxInt, MaxInt+1	TC#21-TC24

Additional Test Cases for Boundary Value Analysis

Test Case ID	(angle)	(distance)	Exp. Result
TF# 21	max int	max int	Error
TC#22	MinInt-1	MinInt-1	Fail
TC#23	MinInt	MinInt	Fail
TC#24	MaxInt	MaxInt	Fail

6.3 hsv2rgb(h, s, v)

a) Equivalence Classes

Parameter: hsv2rgb(h, s, v)	**Equivalence Class** **ID**	**Description**	**Representative**
h	4.1	0 <= h <= 360	80
h	4.a	h < 0	-10
h	4.b	h > 360	490
h	4.c	h is not an int	“this”
s	5.1	0 <= s <= 100	50
s	5.a	s < 0	-12
s	5.b	s > 100	1000
s	5.c	s is not an int	“is a”
v	6.1	0 <= v<= 100	50
v	6.a	v < 0	-9000
v	6.b	v > 100	200
v	6.c	v is not an int	“test case”

b) Derived Test Cases

Test Case ID	**h**	**s**	**v**	**Exp. Result**
TC#17	80	50	50	“output dark shade of green color”
TC#18	-10	-12	-9000	error

TC#19	490	1000	200	error
TC#20	“this”	“is a”	“test case”	error

c) Boundary Value Analysis:

Parameter	Boundary Values	Test Case ID(s)
h	0	TC#17
h	360	TC#17
s	0	TC#17
s	100	TC#17
v	0	TC#17
v	100	TC#17

6.4 draw(radarDisplay, targets, angle, distance, fontRenderer)

a) Equivalence Classes

- Defect detected in this method as putting the pycharm frame value every time we call draw method
- Defect in naming similar names for method and variables
- Defects detected with fontRenderer have font and UI window size in the parameter is not valid; it can be declared but should not be in the method.

Angle and distance didn't need specification-based testing again because of the methods above from target(angle, distance) so only the target array equivalence class was derived and changed the whole method because of the defect (removing radarDisplay (frame) and frontRenderer(font and size) draw method) It should only have target array to draw or have angle and distance directly to draw the dot on the GUI.

Parameter: draw(targets, angle, distance,fontRenderer)	Equivalence Class ID	Description	Representative
targets	7.1	some array	targets[(90,150),(60,110)]
targets	7.a	target is empty	targets[ ]
targets	7.b	targets is not an array	“array”
target	7.c	array malformed	target [{90,120,70},(100)]

b) Derived Test Cases

Test Case ID	targets	angle	distance	Exp. Result
TC#21	targets[(90,150),(60,110)]	90	100	“ draws a dot on the UI at the distance and angle and move its as the robot moves”

c) Boundary Value Analysis:

Parameter	Boundary Values	Test Case ID(s)
targets	max array size	TC#21

6.5 us_map()

Specification-based for us_map is not necessary as the control flow and data flow graph is used to test the function as the function doesn't have parameters to conduct specification-based testing. There are multiple nested loops which can be tested in the sections below to assure the quality of the project.

7. Data-Flow Annotated Control Flow Graphs.

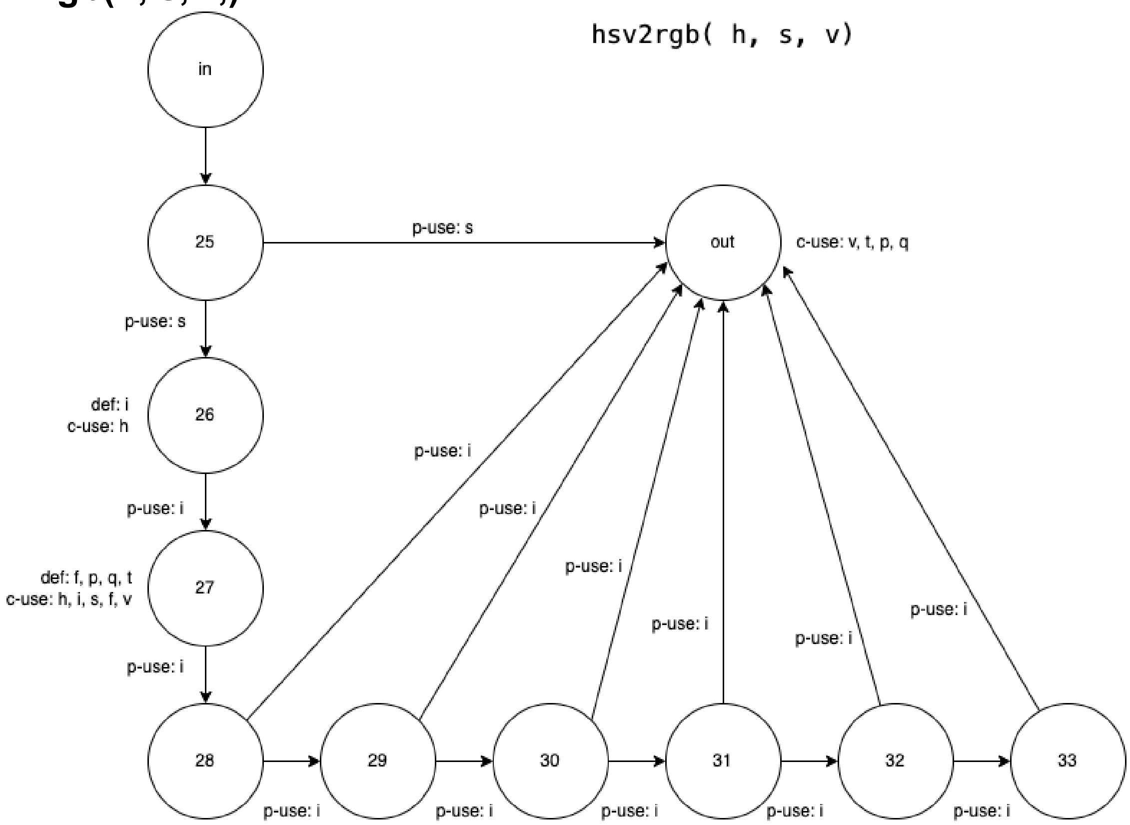
7.1 servo(angle)

servo(angle) is an API call to GoPiGo Servo, we can't perform a Data flow Annotated control flow graph as it was not created as our project but was used to build the project.

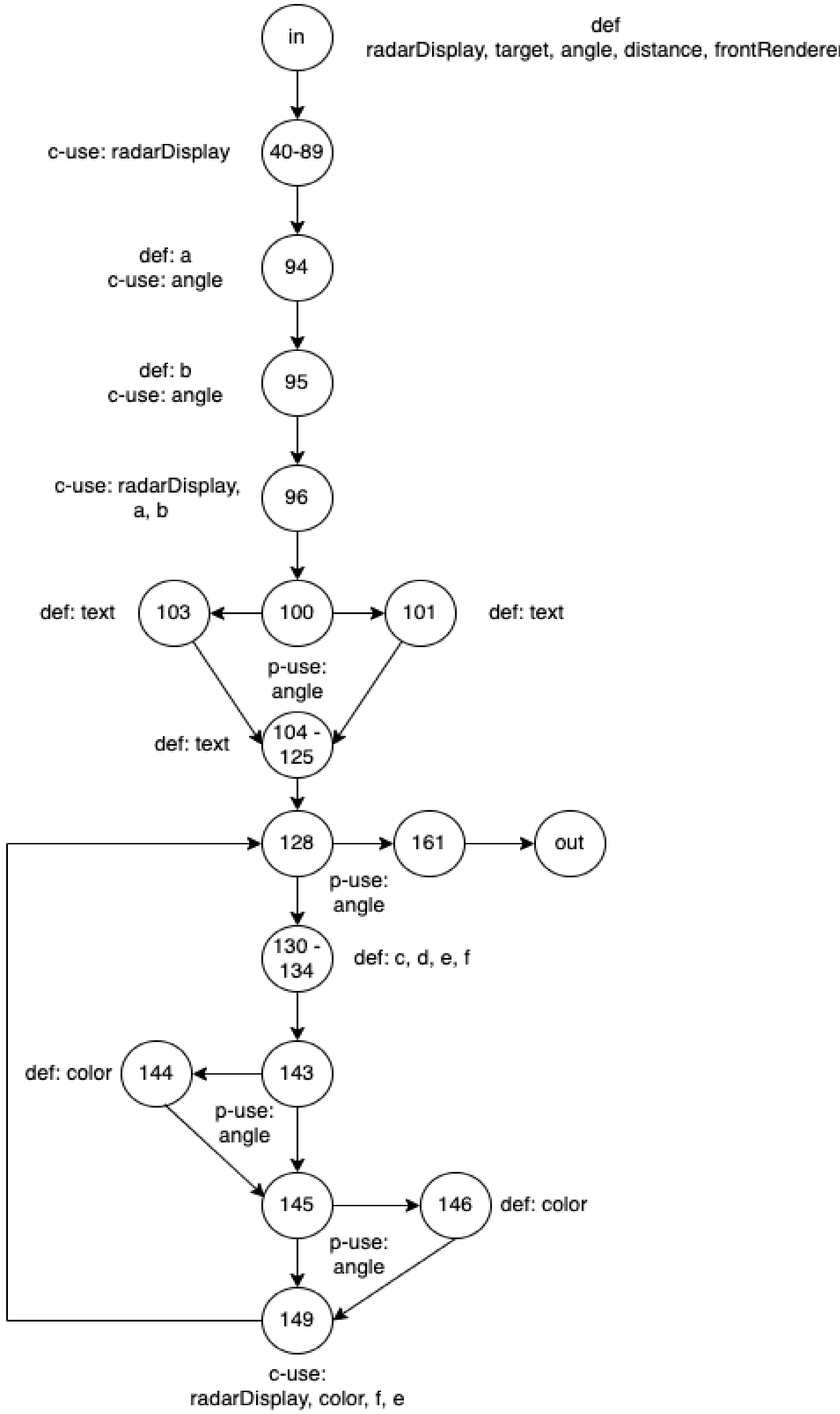
7.2 init(self, angle, distance)

Init(self, angle, distance) method is an instance method with four lines using self. Constructing a Data Flow Annotated control flow graph on a simple and straightforward constructor method isn't necessary.

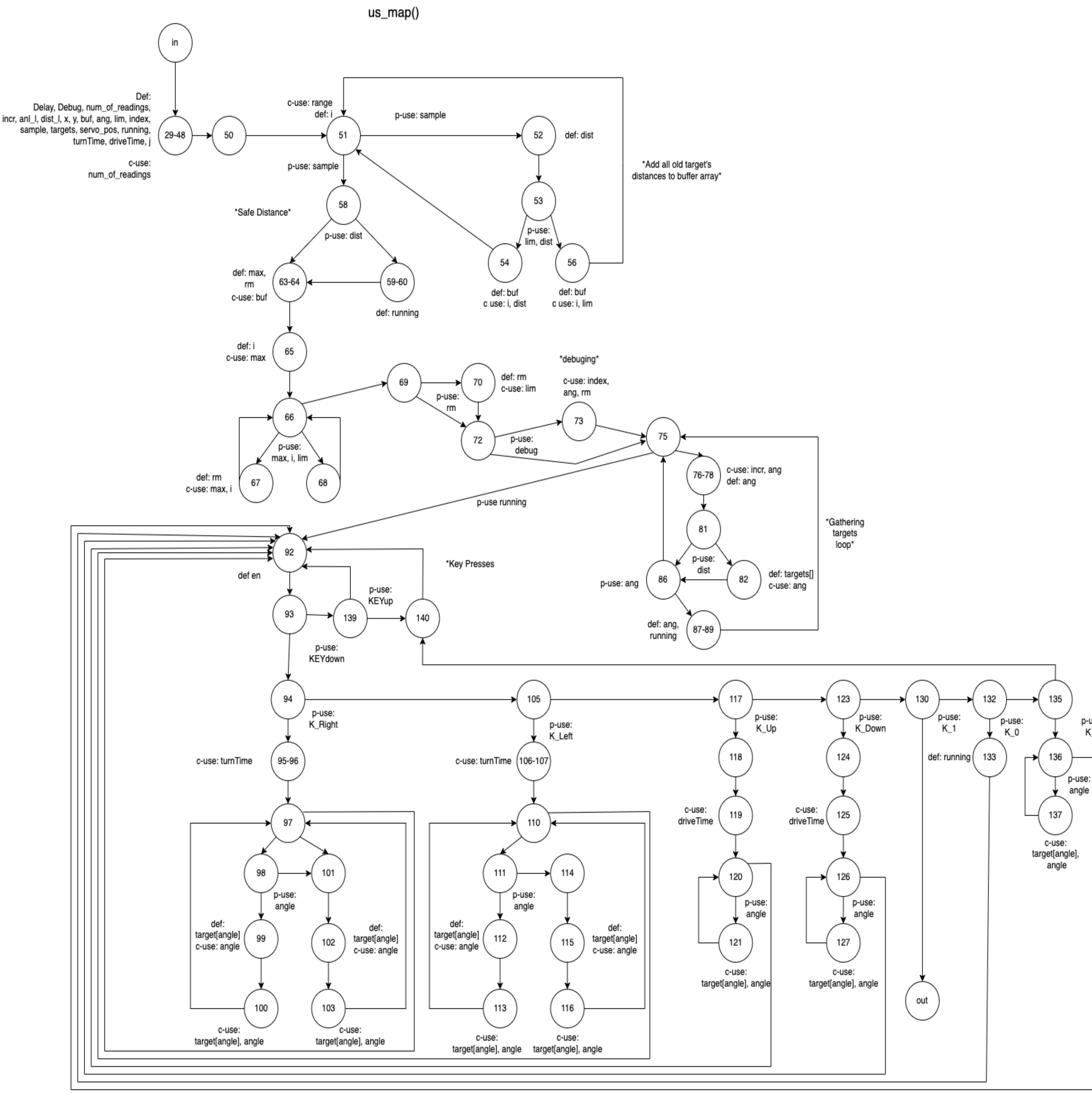
7.3 hsv2rgb(h, s, v,)



7.4 draw(radarDisplay, targets, angle, distance, fontRenderer)



7.5 us_map()



us_map()
in
Def:
Delay, Debug, num_of_readings,
incr, anl_l, dist_l, x, y, buf, ang, lim, index,
sample, targets, servo_pos, running,
turnTime, driveTime, j
c-use:
num_of_readings
29-48
50
c-use: range
def: i
51
p-use: sample
52
def: dist
53
Add all old target's distances to buffer array
p-use: sample
58
Safe Distance
p-use: dist
p-use:
lim, dist
54
56
def: buf
c use: i, dist
def: buf
c use: i, lim
def: max,
rm
c-use: buf
63-64
59-60
def: running
def: i
c-use: max
65
debuging
69
70
def: rm
c-use: lim
p-use:
rm
c-use: index,
ang, rm
73
66
72
p-use:
debug
75
p-use:
max, i, lim
76-78
c-use: incr, ang
def: ang
def: rm
c-use: max, i
67
68
p-use running
81
*Gathering
targets
loop*
92
*Key Presses
def en
p-use:
dist
82
def: targets[]
c-use: ang
p-use: ang
86
93
139
p-use:
KEYup
140
def: ang,
running
87-89
p-use:
KEYdown
94
105
117
123
130
132
135
p-use:
K_Right
p-use:
K_Left
p-use:
K_Up
p-use:
K_Down
p-use:
K_1
p-use:
K_0
p-use:
K_2
c-use: turnTime
95-96
c-use: turnTime
106-107
118
124
def: running
133
136
c-use:
driveTime
119
c-use:
driveTime
125
p-use:
angle
137
97
110
c-use:
target[angle],
angle
98
101
111
114
120
126
p-use:
angle
p-use:
angle
p-use:
angle
p-use:
angle
def:
target[angle]
c-use: angle
99
102
def:
target[angle]
c-use: angle
def:
target[angle]
c-use: angle
112
115
def:
target[angle]
c-use: angle
121
127
c-use:
target[angle], angle
c-use:
target[angle], angle
100
103
113
116
out
c-use:
target[angle], angle
c-use:
target[angle], angle
c-use:
target[angle], angle
c-use:
target[angle], angle

8. Control Flow Based Tests.

8.1 servo(angle)

servo(angle) is an API call to GoPiGo Servo, we can't perform control flow based testing because there isnt Data flow Annotated control flow graph as it was not created as our project but was used to build the project

8.2 init(self, angle, distance)

Init(self, angle, distance) method is an instance method with four lines using self. control flow based is not applicable because there is no Data Flow Annotated control flow graph for a simple and straightforward constructor method isn't necessary.

8.3 hsv2rgb(h, s, v,)

a) C_2-Path coverage testing

Test Case ID	Parameters			Coverage	
	h	s	v	path	%
TC#25	100	0	100	{ in, 25, out }	27.3
TC# 26	300	50	50	{ in,25,26,27,28,out }	55.5
TC# 27	0	100	100	{ in,25,26,27,28,29,out }	63.6
TC# 28	50	100	100	{ in,25,26,27,28,29,30,out }	72.7
TC# 29	100	100	100	{ in,25,26,27,28,29,30,31,out }	81.8
TC# 30	150	100	100	{ in,25,26,27,28,29,30,31,32,out }	90.9
TC# 31	200	100	100	{ in,25,26,27,28,29,30,31,32,33,out }	100

b) C_3-Condition Testing

All permutations of all conditions are already being tested because of the way it was coded, with its array of ‘if’ statements.

8.4 draw(radarDisplay, targets, angle, distance, fontRenderer)

c) C_4-loop testing

Test Case ID	Parameters					Coverage	
	radar Display	targets	angle	distance	fontRenderer	path	%
TC#32	pygame.display.set_model(1400,800)	[120]	120	25	pygame.font.Font(default front20)	{in,40-89,94,95,96,100,103,104-125,128,130-134,143,144,145,149,128,161,out } 18	88.9
TC#33	pygame.display.set_model(1400,800)	[175]	175	55	pygame.font.Font(default front20)	{in,40-89,94,95,96,100,101,104-125,128,130-134,143,145,146,149,128,161,out}	88.9
TC#34	pygame.display.set_model(1400,800)	[120]	120	65	pygame.font.Font(default front20)	{in,40-89,94,95,96,100,103,104-125,128,161,out}	61.1
TC#35	pygame.display.set_model(1400,800)	[170, 175, 180]	170, 175, 180	70, 72, 73	pygame.font.Font(default front20)	{in,40-89,94,95,96,100,101,104-125,128,(130-134,143,145,146,149,128,)^3, 161,out}	88.9

d) C_3-Test (Condition Testing)

Truth Table for C_3-Test

Test Case ID	x	x <= 0	x >= 30	x <= 0|| x >= 30
TT#1	-10	T	F	T
TT#2	40	F	T	T
TT#3	15	F	F	F

8.5 us_map()

d) C_2-Path coverage testing

Test Case ID	Parameters	Coverage	
	no parameters only keyboard inputs	path	**%**
TC#36	(1valid right/s)	{ in,29-48,50,51,52,53,56,51,52,53,54,51,58,63-64,65,66,68,69,70,72,75,76-78,81,82,86,87-89,75,76-78,81,86,75,**92**,93,94,95-96,97,(98,101,102,103,97)^180,(98,99,100,97)^180,92,93,94,105,117,123,130,out }	58%
TC#37	(1debug left/s)	{ in,29-48,50,51,52,53,56,51,52,53,54,51,58,59-60,63-64,65,66,67,66,68,69,72,73,75,76-78,81,82,86,87-89,75,76-78,81,86,75,92,94,105,106-107,110,(111,114,115,116,110)^180,(111,112,113,110)^180,92,93,94,105,117,123,130,out }	63%

TC#38	(1valid f/s)	{ in,29-48,50,51,52,53,56,51,52,53,54,51,58,63-64,65,66,68,69,70,72,75,76-78,81,82,86,87-89,75,76-78,81,86,75,92,93,94,105,117,118,119,120,(121,120)^360,92,93,94,105,117,123,130,out }	52%
TC#39	(1valid b/s)	{ in,29-48,50,51,52,53,56,51,52,53,54,51,58,63-64,65,66,68,69,70,72,75,76-78,81,82,86,87-89,75,76-78,81,86,75,92,93,94,105,117,123,124,125,126,(127,126)^360,92,93,94,105,117,123,130,out }	52%
TC#40	(1valid redraw/restart/stop/)	{ in,29-48,50,51,52,53,56,51,52,53,54,51,58,63-64,65,66,68,69,70,72,75,76-78,81,82,86,87-89,75,76-78,81,86,75,92,93,94,105,117,123,130,132,135,136,(137,136)^360,92,93,94,105,117,123,130,132,133,92,93,94,105,117,123,130,out }	54%
TC#41	(1valid up/s)	{ in,29-48,50,51,52,53,56,51,52,53,54,51,58,63-64,65,66,68,69,70,72,75,76-78,81,82,86,87-89,75,76-78,81,86,75,92,93,94,105,117,123,130,132,135,140,92,93,139,92,93,94,105,117,123,130,out }	52% overall 100%

Paths and coverages were determined by user keystrokes resulting in the robot’s movement.

9. Data Flow Based Tests.

9.1 servo(angle)

servo(angle) is an API call to the GoPiGo servo . We can't perform Data flow based tests as it was not created by use, rather it was used by our project. Going into the API for data flow can make it too complex and diverts from our goal to assure the quality of the Robot Radar then GoPiGo servo API.

9.2 init(self, angle, distance)

Init(self, angle, distance) method is an instance method with four lines using self. performing Data flow based testing on a simple and straightforward method isn't necessary.

9.3 hsv2rgb(h, s, v,)

a) **DEF, C-USE, and P-USE sets**

node n	DEF (n)	C-USE (n)
in	{ h,s,v }	{ }
25	{ }	{ }
26	{ i }	{ h }
27	{ f,p,q,t }	{ h,i,s,f,v }
28	{ }	{ }
29	{ }	{ }
30	{ }	{ }
31	{ }	{ }
32	{ }	{ }

33	{ }	{ }
34	{ }	{ }
35	{ }	{ }
out	{ }	{ v,t,p,q }

edge (n, m)	P-USE (n, m)
{ 25 , out }	{ s }
{ 25 , 26 }	{ s }
{ 28 , out }	{ i }
{ 28 , 29 }	{ i }
{ 29 , out }	{ i }
{ 29 , 30 }	{ i }
{ 30 , out }	{ i }
{ 30 , 31 }	{ i }
{ 31 , out }	{ i }
{ 31 , 32 }	{ i }
{ 32 , out }	{ i }
{ 32 , 33 }	{ i }
{ 33 , out }	{ i }

b) DCU and DPU sets

Variable x	node n	DCU (x, n)	DPU (x, n)
h	in	{ 26,27 }	{ (25 , out),(25 , 26) }
s	in	{ 27 }	{ }
v	in	{ 27,out }	{ }
i	26	{ 27 }	{(28,out),(28,29),(29, out),(29,30),(30,out),(30,31),(31,out)(31,32),(32,out),(32,33),(33, out)}
f	27	{ 27 }	{ }
p	27	{ out }	{ }
q	27	{ out }	{ }
t	27	{ out }	{ }

c) Test Cases for All-Uses Criterion (reused test cases)

Path	Test Case			
	ID	h	s	v
{ in, 25, out }	TC# 25	100	0	100
{ in,25,26,27,28,out }	TC# 26	300	50	50
{ in,25,26,27,28,29,out }	TC#27	0	100	100
{ in,25,26,27,28,29,30,out }	TC#28	50	100	100
{ in,25,26,27,28,29,30,31,out }	TC#29	100	100	100
{ in,25,26,27,28,29,30,31,32,out }	TC#30	150	100	100
{ in,25,26,27,28,29,30,31,32,33,out }	TC#31	200	100	100

9.4 draw(radarDisplay, targets, angle, distance, fontRenderer)

d) DEF, C-USE, and P-USE sets

node n	DEF (n)	C-USE (n)
in	{ radarDisplay, targets, angle, distance, fontRenderer }	{ }
40-89	{ }	{ radarDisplay }
94	{ a }	{ angle }
95	{ b }	{ angle }
96	{ }	{ radarDisplay, a, b }
100	{ }	{ }
101	{ text }	{ }
103	{ text }	{ }
104-125	{ text }	{ }
128	{ }	{ }
130-134	{ c,d,e,f }	{ }
143	{ }	{ }
144	{ color }	{ }
145	{ }	{ }
146	{ color }	{ }
149	{ }	{ radarDisplay, color, f, e }
161	{ }	{ }

edge (n, m)	P-USE (n, m)
{ 100 , 101 }	{ angle }
{ 100 , 103 }	{ angle }
{ 128 , 130 }	{ angle }
{ 128 , 161 }	{ angle }
{ 145 , 146 }	{ angle }
{ 145 , 149 }	{ angle }

e) DCU and DPU sets

Variable x	node n	DCU (x, n)	DPU (x, n)
angle	in	{ }	{(100,101),(100,103),(128,130),(128,161),(145,146),(145,149) }
target	in	{ }	{ }
distance	in	{ }	{ }
fontRenderer	in	{ }	{ }
radarDisplay	in	{ 40-89, 96, 149 }	{ }
a	94	{ }	{ }
b	95	{ }	{ }
c	130-134	{ }	{ }
d	130-134	{ }	{ }
e	130-134	{ 149 }	{ }
f	130-134	{ 149 }	{ }
text	101	{ }	{ }
text	103	{ }	{ }

text	104-125	{ }	{ }
color	144	{ 149 }	{ }
color	146	{ 149 }	{ }

f) Test Cases for All-Uses Criterion

Path	Test Case ID	radarDisplay	targets	angle	distance	fontRenderer
{in,40-89,94,95,96,100,103,104-125,128,130-134,143,144,145,149,128,161,out }	TF# 8	pygame.display.set_model(1400,800)	[120]	120	25	pygame.font.Font(defaultfront20)
{in,40-89,94,95,96,100,101,104-125,128,130-134,143,145,146,149,128,161,out}	TF# 9	pygame.display.set_model(1400,800)	[175]	175	55	pygame.font.Font(defaultfront20)
{in,40-89,94,95,96,100,103,104-125,128,161,out}	TF# 10	pygame.display.set_model(1400,800)	[120]	120	65	pygame.font.Font(defaultfront20)
{in,40-89,94,95,96,100,101,104-125,128,(130-134,143,145,146,149,128,)^3, 161,out}	TF# 11	pygame.display.set_model(1400,800)	[170,175,180]	170, 175, 180	70, 72, 73	pygame.font.Font(defaultfront20)

9.5 us_map()

g) DEF, C-USE, and P-USE sets

node n	DEF (n)	C-USE (n)
in	{ }	{ }
29-48	{Delay,Debug,num_of_readings,incr, anl_l, dist_l, x, y, buf, ang,lim,index,sample,targets, servo_pos, running, turnTime, driveTime,j }	{ num_of_reading }
50	{ }	{ }
51	{ i }	{ range }
52	{ dist }	{ }
53	{ }	{ }
54	{ buf }	{ i, dist }
56	{ buf }	{ i, lim }
58	{ }	{ }
59-60	{ running }	{ }
63-64	{ max, rm }	{ buf }
65	{ i }	{ max }
66	{ }	{ }
67	{ rm }	{ max, i }

68	{ }	{ }
69	{ }	{ }
70	{ rm }	{ lim }
72	{ }	{ }
73	{ }	{ index, ang, rm }
75	{ }	{ }
76-78	{ ang }	{ incr , ang }
81	{ }	{ }
82	{ target[] }	{ ang }
86	{ }	{ }
87-89	{ ang , running }	{ }
92	{ en }	{ }
93	{ }	{ }
94	{ }	{ }
95-96	{ }	{ turnTime }
97	{ }	{ }
98	{ }	{ }

99	{ target[angle] }	{ angle }
100	{ }	{ target[angle] , angle }
101	{ }	{ }
102	{ target[angle] }	{ angle }
103	{ }	{ target[angle],angle }
105	{ }	{ }
106-107	{ }	{ turnTime }
110	{ }	{ }
111	{ }	{ }
112	{ target[angle] }	{ angle }
113	{ }	{ target[angle] , angle }
114	{ }	{ }
115	{ target[angle] }	{ angle }
116	{ }	{ target[angle] , angle }
117	{ }	{ }
118	{ }	{ }
119	{ }	{ driveTime }

120	{ }	{ }
121	{ }	{ target[angle] , angle }
123	{ }	{ }
124	{ }	{ }
125	{ }	{ driveTime }
126	{ }	{ }
127	{ }	{ target[angle] , angle }
130	{ }	{ }
132	{ }	{ }
133	{ running }	{ }
135	{ }	{ }
136	{ }	{ }
137	{ }	{ target[angle], angle }

edge (n, m)	P-USE (n, m)
{ 51, 52 }	{ sample }
{ 51, 58 }	{ sample }

{ 53, 54 }	{ lim dest }
{ 53, 56 }	{ lim, dist }
{ 58,59-60}	{ dist }
{ 58, 63-64 }	{ dist }
{ 66,67 }	{ max,i,lim }
{ 66,68 }	{ max,i,lim }
{ 69,70 }	{ rm }
{ 69 72 }	{ rm }
{ 72, 73 }	{ debug }
{ 72, 75 }	{ debug }
{ 75, 76-78 }	{ running }
{ 75,92 }	{ running}
{ 81,82 }	{ dist }
{ 81,86 }	{ dist }
{ 86, 75 }	{ ang }
{ 86,87-89 }	{ ang }
{ 98, 101 }	{ angle }

{ 98, 99 }	{ angle }
{ 111,112}	{ angle }
{ 111,114}	{ angle }
{ 120,121}	{ angle }
{ 120, 92 }	{ angle }
{ 126,127}	{ angle }
{ 126, 92 }	{ angle }
{ 136,92 }	{ angle }
{ 136,137}	{ angle }

h) DCU and DPU sets

Variable x	node n	DCU (x, n)	DPU (x, n)
Delay	29-48	{ }	{ }
Debug	29-48	{ }	{(72,73),(72,75) }
num_of_readings	29-48	{ 29-48 }	{ }
incr	29-48	{ 76-78 }	{ }
ang_I	29-48	{ }	{ }
dist_I	29-48	{ }	{ }
x	29-48	{ }	{ }
y	29-48	{ }	{ }
buf	29-48	{ 63-64 }	{ }

angel	29-48	{ 99,100,102,103,112, 115,116,121,127,137 }	{(98,101),(98,99),(111,112),(111,114),(120,121),(120,92),(126,127),(126,92),(136,92),(136,137)}
lim	29-48	{ 56,70 }	{ (53,54), (53,56), (66,67) ,(66,68) }
index	29-48	{ 73 }	{ }
sample	29-48	{ }	{ (51,52),(51,58) }
targets	29-48	{ }	{ }
servo_pos	29-48	{ }	{ }
running	29-48	{ }	{ (75,76-78), (75-792,)}
turnTime	29-48	{ 95-96,106-107 }	{ }
driveTime	29-48	{ 119 ,125 }	{ }
j	29-48	{ }	{ }
i	51	{ 54,56, 67}	{(66,67)(66,68) }
dist	52	{ 54,}	{ (53,54),(53,56),(58,59-60)(58,63-64),(81,82),(81,86) }
buf	54	{ 63-64 }	{ }
buf	56	{ 63-64 }	{ }
running	59-60	{ }	{ (75,76-78), (75-792,)}
max	63-64	{ 65,67 }	{(66,67)(66,68) }
rm	63-64	{ 73 }	{(69,70),(69,72) }
i	65	{ 54,56, 67 }	{(66,67)(66,68) }

rm	67	{ 73 }	{(69,70),(69,72) }
rm	70	{ 73 }	{(69,70),(69,72)}
angle	76-78	{ 99,100,102,103,112, 115,116,121,127,137 }	{(98,101),(98,99),(111,112),(111,114),(120,121),(120,92),(126,127),(126,92),(136,92),(136,137)}
target[]	82	{ 100,103,116,121,127,137 }	{ }
ang	87-89	{ 73,76-78,82 }	{(86,75),(86,87-89)}
running	87-89	{}	{ (75,76-78), (75-792,)}
target[]	99	{ 100,103,116,121,127,137 }	{ }
target[]	102	{ 100,103,116,121,127,137 }	{ }
target[]	112	{ 100,103,116,121,127,137 }	{ }
target[]	115	{ 100,103,116,121,127,137 }	1. { }
running	133	{ }	{ (75,76-78), (75-792,)}

i) Test Cases for All-Uses Criterion

Can't perform all use criteria because the method does not have any parameter and this being the main method for the program . The flow of data through the method is more like ‘the sensor reads for the data to show on the GUI to the user’ and the ‘user input’ is the other data required for the robot to move around. data is created when the program runs.

10. Class Test Strategy

The Modality of the classes in Robot Radar are mostly modal. Allowed method calls depend on current attributes values and the sequences of previous method calls. We had to design it in such a fashion in order for certain actions to not occur before other ones. (For example; Driving only after a full scan is performed by the radar)

10.1 Method Scope Test.

The following table explains and shows each class in the robotradar its modality with the explanation of why it modaled that way.

Class	Methods	Modality	Explanation
color.py	-	**Does Not Contain Methods**	Method is an enumeration
display.py	hsv2rgb(h, s, v) draw(radarDisplay, targets, angle, distance, fontRenderer)	**Modal**	Allowed method calls depend on current attributes values and the sequences of previous method calls The order of the method calls are very important for setting up the display properly. Certain aspects of the interface have to be made before the addition of other artifacts onto the display window, (targets, angle, sweeping radar)
radar.py	us_map()	**Modal**	Allowed method calls depend on current attributes values and the sequences of previous method calls. The sequence in which the methods inside the class are referenced are very dependent on what occurs prior to said call, as it determines how and when the robot moves and uses the servo. We made the class like this to ensure that a safety distance check was performed before the robot could start driving again, meaning that certain functions, like movement, are only allowed to be called after a radar sweep.
gopigo.py	servo(angle)	**Modal**	Used for relaying instructions to the servo to have the angle of the servo, and ultimately the ultrasonic sensor, move. its an API called used from GoPiGo
main.py	-	**Does Not Contain Methods**	Only used to start the program, does not contain any other methods
target.py	__init__(self, angle, distance)	**Non-Modal**	Only a Constructor, used for marking individual targets.

10.1a Category Partition Test

I. servo(angle).

Category partition testing extending Equivalence class testing **servo(angle)**

The following table show primary and secondary function of the method servo(angle)

	Function
Primary	Used for relaying instructions to the servo to have the angle of the servo, and ultimately the ultrasonic sensor, move. its an API called used from GoPiGo
Secondary	API supply usmap() radar the ability to move sevor for the radar to scan for objects

Category partition testing cant be derived because the method calls gopigo API to rotate the servo.

II. init(self, angle, distance).

Category partition testing extending Equivalence class testing **init (angle, distance)**

The following table show primary and secondary function of the method __init__(self, angle, distance)

	Function
Primary	Used for constructing Target Objects (~ Angle, Distance)
Secondary	creates multiples objects

Category partition testing can be derived because its a constructor class

III. hsv2rgb(h, s, v).

Category partition testing extending Equivalence class testing **hsv2rgb(h, s, v)**

The following table show primary and secondary function of the method hsv2rgb

	Function
Primary	Unused in RobotRader for change color on depending the distance of object
Secondary	does not have a secondary function because the methos isnt called by the program

Category partition testing cannot be derived because the function does not have a secondary function in the program as it has not been used; the fix would be to delete the method and free the unused code from the program.

IV. draw(radarDisplay, targets, angle, distance, fontRenderer)

Category partition testing extending Equivalence class testing
draw(radarDisplay, targets, angle, distance, fontRenderer)

The following table show primary and secondary function of the method draw(radarDisplay, targets, angle, distance, fontRenderer)

	Function
Primary	Initializes the pygame's display window, Draws the UI in the pygame's screen, Loops through all target objects and draws them on the display,Displays the servo's current angle
Secondary	supply radar with the init frame and object to display on use interface

Category partition testing

Interface parameters,	ID	Description	Representative
targets	7.1	some array	targets[(90,150),(60,110)]
targets	7.a	target is empty	targets[ ]
targets	7.b	targets is not an array	"array"
target	7.c	array malformed	target [{90,120,70},(100)]

Derived Test Cases

Test Case ID	targets	angle	distance	Exp. Result
TC#21	targets[(90,150),(60,110)]	90	100	" draws a dot on the UI at the distance and angle and move its as the robot moves"

Boundary Value Analysis:

Parameter	Boundary Values	Test Case ID(s)
targets	max array size	TC#21

V. us_map()

Category partition testing extending Equivalence class testing **us_map()**

The following table show primary and secondary function of the method us_map

	Function
Primary	Function: Moves the Servo to 0, Checks if objects are too close, Waits for inputs to start moving
Secondary	takes user input to move the robot

Deriving Category Partition Testing Values for us_map()

Interface parameters,	ID	Description	Representative
Delay	8.1	some double.	0.2
Delay	8.a	NOT a double.	“bark”
Debug	9.1	some int	1
Debug	9.a	not an int	“zero”
Num_of_readings	10.1	some int	45
Num_of_readings	10.a	not an int	“fortyfive”
Incr	11.1	some int	2
Incr	11.a	not an int	“circles”

buf	12.1	some int	40
buf	12.a	not an int	“buffer”
ang	13.1	some int	90
ang	13.a	not an int	“angle”
lim	14.1	some int	1000
lim	14.a	not an int	“limit”
index	15.1	some int	0
index	15.a	not an int	“index”
sample	16.1	some int	2
sample	16.a	not an int	“sample”
targets	17.1	some array	targets[(90,150),(60,110)]
targets	17.a 17.b 17.c	target is empty targets is not an array array malformed	targets[ ] “array” target [{90,120,70},(100)]
running	18.1	some int	1
running	18.a	not an double	“running”
turnTime	19.1	some double	0.305
turnTime	19.a	not an double	“turnTime”
driveTIme	20.1	some double	0.305
driveTIme	20.a	not an double	“driveTIme”

Derived Test Cases

Test Case ID	Variable	Input value	Exp. Result
TC#42	Delay	0.02 2	time delay by 0.02seconds time delay by 2 seconds
TC#43	Debug	0 1	does not print index, ang, rm print index, ang, rm
TC#44	Num_of_readings	45 90	“incrementing 45 degree” “incrementing 90 degree”
TC#45	Incr	2 5	“incrementing 2 cycle” “incrementing 5 cycle”
TC#46	buf	0 2	buffering by 0 buffering by 2
TC#47	ang	15 30	angle 15 degree angle 30 degree
TC#48	lim	1000 100	1000 centimeter scan radar 100 centimeter scan radar
TC#49	index	0 1	does not print index, print index, ang
TC#50	sample	0 2	scan 0 rounds scan 2 rounds
TC#51	targets	[(90,150),(60, 110)]	draws a dot on the UI at the distance and angle and move its as the robot moves
TC#52	running	0 1	stop running keeps running
TC#53	turnTime	0.305 0.500	speed 0.305 speed 0.500
TC#54	driveTIme	0.305 0.500	speed 0.305 speed 0.500

Tests of individual method are carried out in:
Section 5) Specification based testing.

10.1b Source-Code Test.

source-code-based testing is carried out in:
Section 7) Control Flow based testing.

- hsv2rgb(h, s, v,) - C2-Path coverage testing
- draw(radarDisplay, targets, angle, distance, fontRenderer) - C4-loop testing
- us_map() - C2-Path coverage testing
- hsv2rgb(h, s, v,) - All-Uses Criterion
- draw(radarDisplay, targets, angle, distance, fontRenderer) - All-Uses Criterion

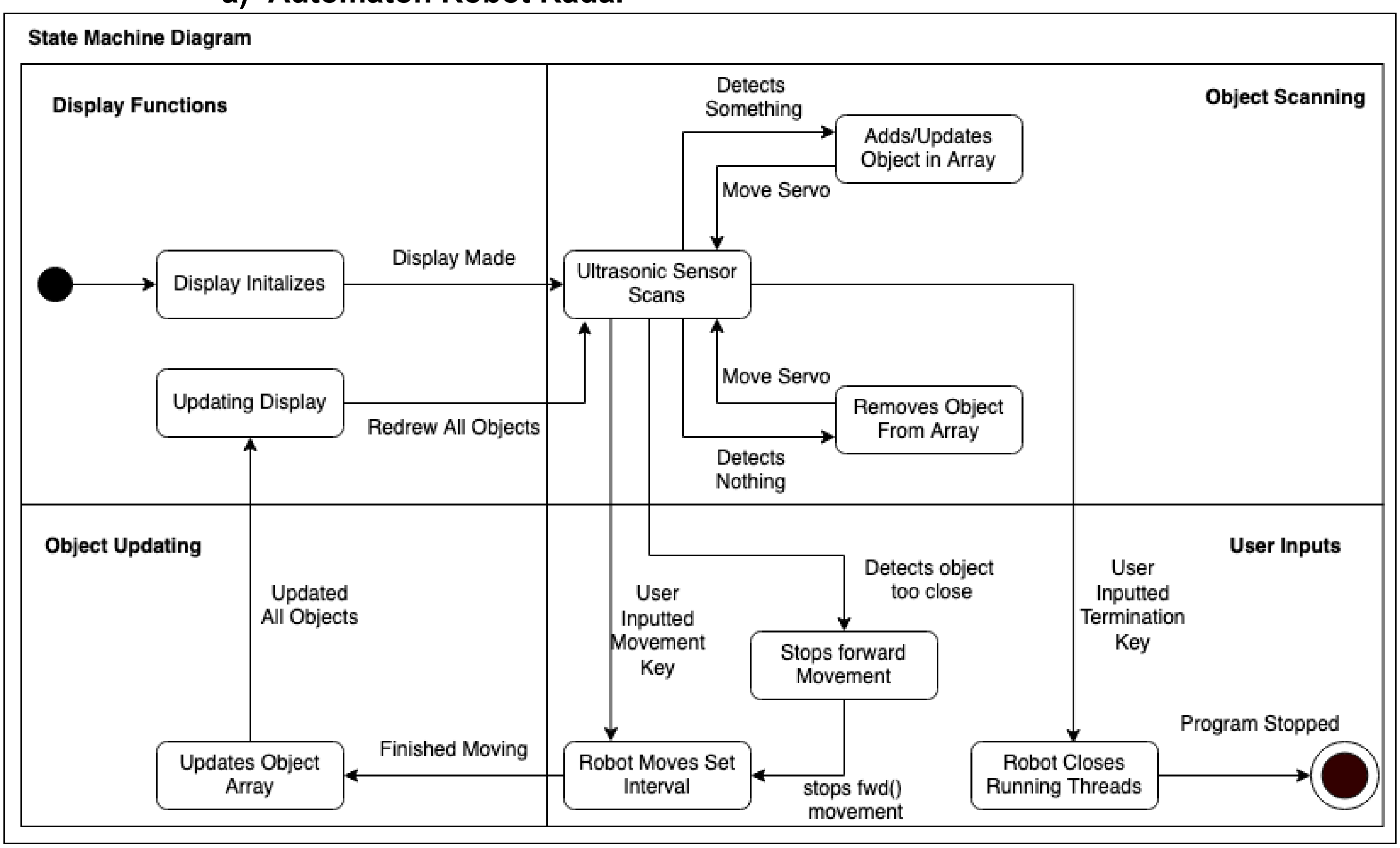
10.1c Polymorphism Test

No polymorphism for robot radar since there are no inheritances inside of the classes.

10.2 Class Scope Test.

Class Scope testing the call order of methods depends on the object state we are using state based testing approach derving sequences to be tested according to the event coverage criterion.

a) Automaton Robot Radar



1) Transition Matrix:

State		Event / Transition										
ID	state	Display Made	Detects Something	User Inputted Termination Key	Move Servo	Detects Nothing	Detects object too close	User Inputted Movement Key	Finished Moving	Updated All Objects	Redrew All Objects	stops fwd() movement
1	Display Initializes	2										
2	Ultrasonic Sensor Scans		3			5	7	6				
3	Adds/ Updates Object in Array				2							
4	Updating Display										2	
5	Removes Object From Array				2							
6	Robot Moves Set Interval								8			
7	Stops forward Movement											6
8	Updates Object Array									4		
9	Robot Closes Running Threads	-	-	-	-	-	-	-	-	-	-	-

2) Coverage Criterion: All Events

Test Case ID	Event Vector	Exp. End state	Coverage Covered Events	%
TC55	Display Made (2), User Inputted Movement Key (6), Finished Moving (8), Updated All Objects (4), Redrew all Objects (2), User Inputted Termination Key (9).	Robot Closes Running Thread	Display Initialization, Ultrasonic Sensor Scan, Adds/Updates Object in Arrays, Ultrasonic Sensor Scan, Stop Forward Object Movement, Robot Move set Interval, Update Object Array, Update Display, Ultrasonic Sensor Scan, Robot Closes Running Thread	50%
TC56	Display Made (2), Detects Something (3), Moves Servo (2), Detects Object Too Close (7), Stop Forward Movement (6), Finished Moving (8), Updated All Objects (4), Redrew all Objects (2), User Inputted Termination Key (9).	Robot Closes Running Thread	Display Initialization, Ultrasonic Sensor Scan, Robot Movement Set Interval, Update Object Array, Update Display, Ultrasonic Sensor Scan, Robot Closes Running Thread	75%
TC57	Display Made (2), Detects Something (3), Moves Servo (2), Detect Nothing (5), Moves Servo (2), User Inputted Movement Key (6), Finished Moving (8), Updated All Objects (4), Redrew all Objects (2), User Inputted Termination Key (9).	Robot Closes Running Thread	Display Initialization, Ultrasonic Sensor Scan, Adds/Updates Object in Arrays, Ultrasonic Sensor Scan, Remove Object from Array, Ultrasonic Sensor Scan, Robot Move set Interval, Update Object Array, Update Display, Ultrasonic Sensor Scan, Robot Closes Running Thread	75%

b) Flattened Class Scope Test and Class Interaction Test.

No Inheritance for robot radar

11. Quality Tools

The following lists the tools to be used to support quality management implementation and the purpose or use of the tool.

IDE - Pycharm
Testing Software - Pytest

12. Quality Control and Assurance Problem Reporting Plan

Quality was managed rigorously, The project assurance will monitor quality and report exceptions using the following table logs to itemize, document, and track closure items reported through quality management activities.

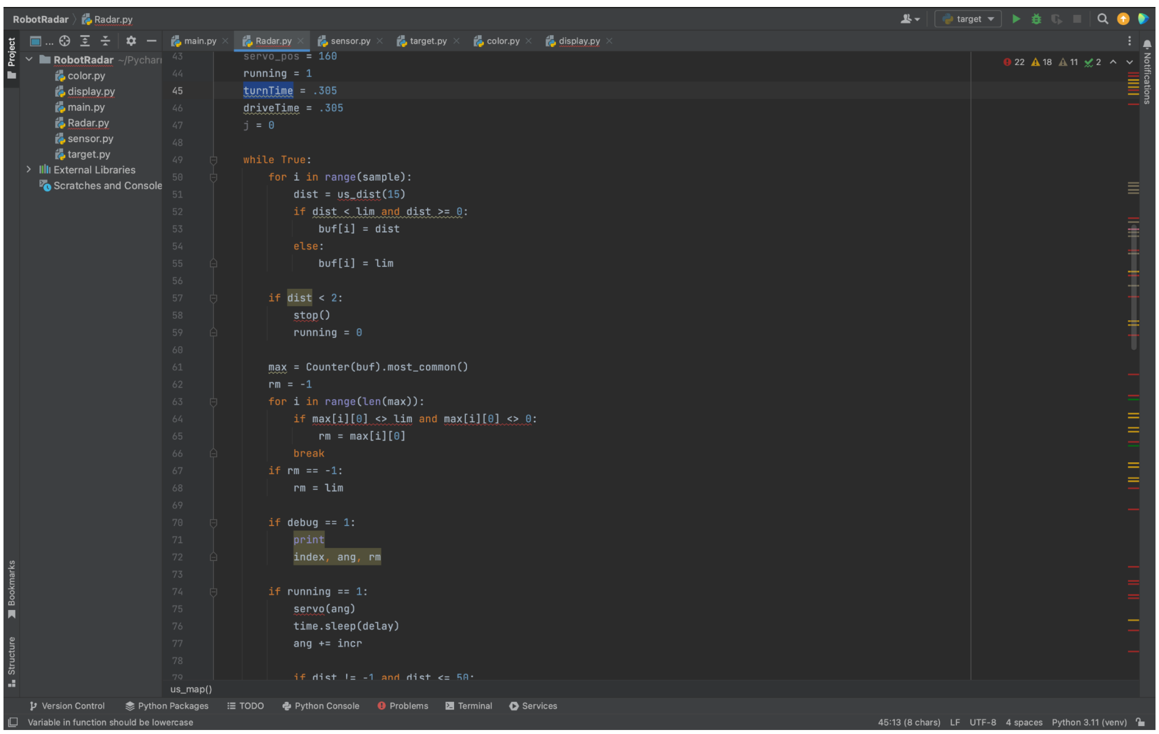
Quality Control Log

Exception ID Number	Review Date	Findings	Resolution	Error/Fault/Failure
QC-Exc-1	10/5/2022	We found a defect in method draw has parameter radardisplay which passes it every time while its a pygame GUI frame size	remove the radarDisplay from the method as it does not need to be changed every time draw is called while it can be called in display class	Fault: Poor code implementation leading to inefficient runtime execution.
QC-Exc-2	10/5/2022	We found draw method parameter naming the variables similar to other methods name and variable same (This might be confusing for a person if they didn't have an idea of what happening in the coding and more commenting is require)	Resolution have current servo_angle, current_angle or current_distance or object_distance (name the variable and method a little difference and specific names to reduce confusion for the person maintaining the code/ software	Fault: Poor naming conventions may lead to bad readability and confusion later on.
QC-Exc-3	10/6/2022	We found fontRenderer in the method parameter which is just the font name and size which should be declared in the class but not included in the method.	Removed the fontRenderer from the method as it does not need to be changed every time draw is called.	Fault: Poor code implementation leading to inefficient runtime execution.
QC-Exc-4	11/1/2022	Extensively long methods and nested loops which might be hard to track the out	break it down in multiple methods and trying to make the loops a little	Fault: Confusing and oddly written loops and if

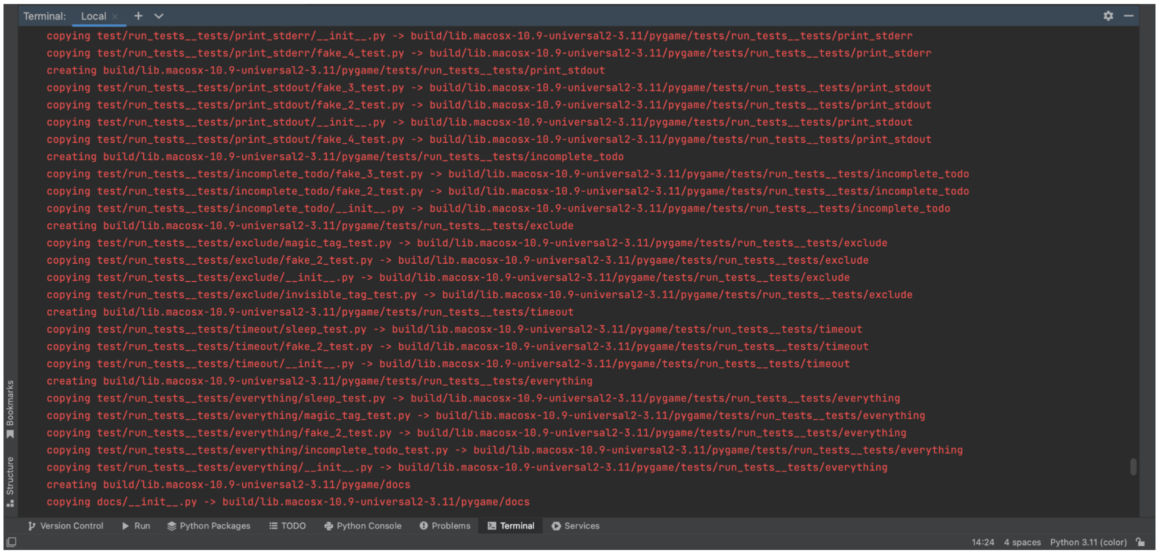
		comes (movement method, servo movement method, updating screen in a method all concurrently)	less nested.(tracking and testing will be easier with less nested loops)	statements may lead to bad readability and confusion later on.
QC-Exc-5	11/3/2022	Found a useless pair of ‘if’ statements, the second ‘if’ should just be an ‘else’ because it is always the case that if the first ‘if’ isn't true the second if is	changed the pair of ‘if’ statements to ‘else’ (in draw method line 143-145)	Fault: Poor code implementation for loop which can be if else then making two branch loop again
QC-Exc-6	11/3/2022	unused method in draw hsv2rgb	the method does some weird bit changing to change color picked from stack overflow found even playing with it around.	Error: 2 people were working on separate branches of the code and made essentially duplicate methods
QC-Exc-7	11/30/2022	found unused variable in radar.py variable ang_l, dist_l, x, y, j and pos	delete unused variable	Error: Unnecessary code
QC-Exc-8	11/30/2022	found int variable used as boolean 0 and 1 in radar.py variable names index debug and running	use boolean variable for true or false	Fault: Poor code implementation and use of variable having a boolean is easier for logic operation decisions
QC-Exc-9	12/1/2022	We have noticed that the way the robot turns, every target will slowly be incremented incorrectly due to the turning radius of the robot	Take into account where the servo is on the robot	Error: Didn't realize that the radar offset changed the readings significantly. Fault: The robot does not take into consideration that the radar isn't in the middle of the robot

				Failure: The targets will slowly become increasingly more incorrect
QC-Exc-10	12/2/22	Found an unused import, pyCamera	Remove the unused import	Error: Useless import leading to inefficient runtime execution.

13. Summary pyUnit screenshot



Above we can see multiple problems that the API are being called but aren't compatible for laptops as they specifically need to be run on Gopigo robots as they aren't supported on normal machines.



When setting up the PyTest there were a few problems that were encountered. When attempting to get the code running without the actual robot, more than half of the tests we would want to run would be impossible as they are testing the functionality of the physical robot interacting with targets in the real world; meaning that we would have to remove almost all our current API's and scaffold in all our targets and remove any trace of this even being a moving robot. PyTest would essentially be only testing the functionality of the GUI and when that is the case, PyTest does not seem appropriate to test a GUI.

14. Results

Keyboard user input test cases and results

Input Key	Expected output	Results
Up	robot goes forward	Success
Left	robot turns left	Success
Down	robot goes backward	Success
Right	robot turns right	Success
0	radar sweep	Success
1	program exits	Success

Robot sensor results

Input Key	Expected output	Results
Sensor data to display	print angle of the object	Success

Colored GUI on sensor reading

Input Key	Expected output	Results
Object to radar/map	Draw a red,yellow, green dot [Red- Object >15] [Yellow- 15<Object >35] [Green- Object >35]	Success
Robot Movement to object	change dot according to direction of the robot movement	Success
Sensor Update	update dot location	Success

GUI

Input Key	Expected output	Results
sensor/servo scanning	Use the servo angle movement to print Radar grid	Success
Initial Sensor data to radar	print dot on the radar grid	Success
Movement after scanning object	Robot keeps track of object and obstacle movement	Success

Conclusion:
We have a sufficient amount of test cases to test this using PyUnit but we can't perform them due to the machine incompatibility and we can still justify the that most of the test which are expected to pass will pass and those expected to fail will fail as we had tested the robot during CSC 380 last few weeks of class. The limitations and constraints for these projects were a little too much as we were making a program for a different machine. For example while working in CSC 380 we had to rewrite the code on Gopigo to run it and now we can't perform those tests. Justifying our Quality property about 90% of tests will pass for correctness and efficiency and about 10% of features are missing one of them is camera live stream from Gopigo. further below is the code of the unit test for each file

15. Unit test with MagicMock

15.1 File name colors.py

```
import unittest

class TestColors(unittest.TestCase):
    def test_white_color(self):
        self.assertEqual(white, (255, 255, 255))

    def test_black_color(self):
        self.assertEqual(black, (0, 0, 0))

    def test_red_color(self):
        self.assertEqual(red, (255, 0, 0))

    def test_orange_color(self):
        self.assertEqual(orange, (255, 165, 0))

    def test_green_color(self):
        self.assertEqual(green, (0, 255, 0))

    def test_transparent_red_colors(self):
        self.assertEqual(red1L, (255, 26, 26))
        self.assertEqual(red2L, (255, 51, 51))
        self.assertEqual(red3L, (255, 77, 77))
        self.assertEqual(red4L, (255, 102, 102))
        self.assertEqual(red5L, (255, 128, 128))
        self.assertEqual(red6L, (255, 153, 153))

    def test_green_color(self):
        self.assertEqual(green, (0, 255, 0))

    def test_yellow_color(self):
        self.assertEqual(yellow, (255, 255, 0))

    def test_blue_color(self):
        self.assertEqual(blue, (0, 0, 255))

if __name__ == '__main__':
    unittest.main()
```

15.2 display.py

```
import unittest
from unittest.mock import MagicMock
import colors
import colorsys
import pygame
import picamera
import math
import time
from gopigo import *
import sys
from collections import Counter
import io
from radar import *

class TestRadar(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        cls.colors = MagicMock()
        cls.colorsys = MagicMock()
        cls.pygame = MagicMock()
        cls.picamera = MagicMock()
        cls.math = MagicMock()
        cls.time = MagicMock()
        cls.gopigo = MagicMock()
        cls.sys = MagicMock()
        cls.Counter = MagicMock()
        cls.io = MagicMock()
        cls.radar = MagicMock()

    def test_hsv2rgb(self):
        hsv2rgb = MagicMock()
        self.assertTrue(hsv2rgb(0.5, 0.5, 0.5))

        def test_hsv3rgb(self):
        hsv3rgb = MagicMock()
        hsv3rgb.return_value = (128, 128, 128)
        self.assertEqual(hsv3rgb(0.5, 0.5, 0.5), (128, 128, 128))

    def test_draw(self):
        draw = MagicMock()
        radarDisplay = MagicMock()
        targets = MagicMock()
```

```
        angle = MagicMock()
        distance = MagicMock()
        fontRenderer = MagicMock()
        draw(radarDisplay, targets, angle, distance, fontRenderer)
        draw.assert_called()

    def test_math_sin(self):
        math_sin = MagicMock()
       math_sin.return_value = 0.5
        self.assertEqual(math_sin(0.5), 0.5)

    def test_math_cos(self):
        math_cos = MagicMock()
        math_cos.return_value = 0.5
        self.assertEqual(math_cos(0.5), 0.5)

    def test_pygame_draw_circle(self):
        pygame_draw_circle = MagicMock()
        pygame_draw_circle.assert_called()

    def test_pygame_draw_rect(self):
        pygame_draw_rect = MagicMock()
        pygame_draw_rect.assert_called()

    def test_pygame_draw_line(self):
        pygame_draw_line = MagicMock()
        pygame_draw_line.assert_called()

if __name__ == '__main__':
    unittest.main()
```

15.3 main.py

```
import unittest
from unittest.mock import MagicMock

class TestRadar(unittest.TestCase):
    def test_gopigo_import(self):
        gopigo = MagicMock()
        gopigo.assert_not_called()

    def test_print_robot_radar(self):
        print_robot_radar = MagicMock()
        print_robot_radar("Robot Radar")
        print_robot_radar.assert_called_once_with("Robot Radar")
```

```
    def test_print_loading(self):
        print_loading = MagicMock()
        print_loading("Loading......................")
        print_loading.assert_called_once_with("Loading......................")

    def test_while_loop(self):
        radar_import = MagicMock()
        while_loop = MagicMock()
        while_loop(True)
        while_loop.assert_called()

if __name__ == '__main__':
    unittest.main()
```

15.4 radar.py

```
import unittest
from unittest.mock import MagicMock
import pygame
import io
import math
import time
import colors
import sys
from target import *
from display import *
from gopigo import *
from collections import Counter

class TestRadar(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        cls.pygame = MagicMock()
        cls.picamera = MagicMock()
        cls.io = MagicMock()
        cls.math = MagicMock()
        cls.time = MagicMock()
        cls.colors = MagicMock()
        cls.sys = MagicMock()
        cls.target = MagicMock()
        cls.display = MagicMock()
```

```
        cls.gopigo = MagicMock()

    def test_us_map(self):
        us_map = MagicMock()
        self.assertTrue(us_map())

    def test_stop(self):
        stop = MagicMock()
        stop.assert_called()

    def test_right(self):
        right = MagicMock()
        right.assert_called()

    def test_left(self):
        left = MagicMock()
        left.assert_called()

    def test_fwd(self):
        fwd = MagicMock()
        fwd.assert_called()

    def test_servo_angle(self):
        servo = MagicMock()
        servo.return_value = 180
        self.assertEqual(servo(180),180)

    def test_color(self):
        color = MagicMock()
        color.assert_called()

if __name__ == '__main__':
    unittest.main()
```

15.5 sensorTest.py

```
import unittest
from unittest.mock import MagicMock
from gopigo import *
import time

class TestGoPiGo(unittest.TestCase):
    def test_stop_on_obstacle(self):
```

```
        distance_to_stop = 20  # Distance from obstacle where the GoPiGo should
stop
        raw_input = MagicMock(return_value='')  # simulate user input
        us_dist = MagicMock(side_effect=[10, 15, 20, 25, 30])  # simulate sensor
measurements

        fwd()  # Start moving

        while True:
            dist = us_dist(15)  # Find the distance of the object in front
            self.assertLess(dist, distance_to_stop) # Check whether it stops at 20
            stop()  # Stop the GoPiGo
            break

            time.sleep(.1)

    if __name__ == '__main__':
        unittest.main()
```

15.6 Target.py

```
import unittest
import time
import colors
from gopigo import *

class TestTarget(unittest.TestCase):
    def test_init(self):
        target = Target(45, 30)
        self.assertEqual(target.angle, 45)
        self.assertEqual(target.distance, 30)
        self.assertAlmostEqual(target.time, time.time(), delta=1e-5)
        self.assertEqual(target.color, ())

    def test_set_color(self):
        target = Target(0, 0)
        target.color = (255, 0, 0)
        self.assertEqual(target.color, (255, 0, 0))
    def test_distance_setter(self):
        target = Target(0, 0)
        target.distance = 15
        self.assertEqual(target.distance, 15)

if __name__ == '__main__':
    unittest.main()
```