FollowThrough: Test Report - Revision 1

Djordje Petrovic
1306258
petrod@mcmaster.ca

George Plukov
1316246
plukovga@mcmaster.ca

Zayan Imtiaz
1152665
imtiaz3@mcmaster.ca

Philip Habib
1310455
habibp@mcmaster.ca

April 2017

CONTENTS

1 INTRODUCTION 3
- 1.1 Revision Table 3
- 1.2 List of Tables 3
- 1.3 Purpose of the Document 3
- 1.4 Scope of the Testing 3
- 1.5 Organisation of the Document 4

2 UNIT TESTING 5
- 2.1 Ball_Tracker module 5
  - 2.1.1 Ball_Tracker.begin_capture() 5
  - 2.1.2 Ball_Tracker.angle(x1, y1, x2, y2) 6
  - 2.1.3 Ball_Tracker.direction(pts, direction, frames) 6
- 2.2 Networking module 7
  - 2.2.1 Networking_Python.post(url) 8
  - 2.2.2 Networking_Hardware.post(url) 8

3 HARDWARE TESTING 10
- 3.1 Basket Tracker 10
- 3.2 Distance Sensor 10
- 3.3 Wifi Module 10
- 3.4 Raspberry Pi 10

4 TRACEABILITY 12
- 4.1 Requirements 12
- 4.2 Modules 12

5 CHANGES AFTER TESTING 13

1

INTRODUCTION

1.1 REVISION TABLE

Revision Number	**Date**	**Description**
1	March 25th 2017	Starting the document. 2
April 8th 2017	Making minor changes.	

1.2 LIST OF TABLES

Table Number	**Name**
2.1.1.1	`Ball_Tracker Test Inputs`
2.1.1.2	`Ball_Tracker Test Results`
2.1.2.1	`Ball_Tracker.angle Test Inputs`
2.1.2.2	`Ball_Tracker.angle Test Results`
2.2.1.1	`Networking_Python.post Test Inputs`
2.2.1.2	`Networking_Python.post Test Results`
2.2.2.1	`Networking_Hardware.post Test Inputs`
2.2.2.2	`Networking_Hardware.post Test Results`
4.1	`Requirements`
4.2	`Modules`

1.3 PURPOSE OF THE DOCUMENT

This document is meant to showcase our testing of the FollowThrough- virtual basketball coach product. This document has been laid out so that it covers the three sections where testing will take place.

1.4 SCOPE OF THE TESTING

The following tests will all be conducted on various edge cases and general cases to ensure that the product works on all levels prior to shipping. Most of the testing will be done as a case by case basis, as unit testing is mostly inapplicable with this product. The cases will cover different angles and various hardware placements while the unit testing will be done on a per function basis. For example the algorithm that calculates entry angle can and will be unit tested, whereas the angle at which the camera is positioned can only be tested by case.

These tests are to be a tool to discover the limitations of the FollowThrough- product, not just test what we already know to be true. A general understanding of the product and some understanding of computer science content is assumed.

1.5 ORGANISATION OF THE DOCUMENT

The first section will cover unit testing. This section pertains to testing the software's algorithms and networking aspects of the website. The second section will cover the testing of the hardware, which is mostly in the form of scenario testing. The third section will cover the traceability of the tests in correlation to the requirements. Finally the fourth section will discuss the potential changes after the testing.

2

UNIT TESTING

2.1 BALL_TRACKER MODULE

The following will be manual testing of the functions in the Ball_Tracker python module. The Ball_Tracker class in the Ball_Tracker module is instantiated as follows:

```
Ball_Tracker(is_live, video_path)
```

where *is_live* is a boolean specifying whether the user is streaming a video or recording live and *video_path* is a string specifying the path of the video, which of course is only required if *is_live* is set to *False*.

2.1.1 *Ball_Tracker.begin_capture()*

The function begin_capture() can capture from both the a camera device mounted on the user's computer or a supplied video, depending on the initiation of the class Ball_Tracker. The method will be called as follows:

```
Ball_Tracker(is_live, video_path).begin_capture()
```

2.1.1.1 *Test Inputs*

Test No.	**Input Params**
1	*is_live* = True, *video_path* = None
2	*is_live* = True, *video_path* = 'test_videos/FT_make.MOV'
3	*is_live* = False, *video_path* = 'test_videos/FT_make.MOV'
4	*is_live* = False, *video_path* = 'test_videos/FT_miss.MOV'
5	*is_live* = False, *video_path* = None
6	*is_live* = False, *video_path* = ''

2.1.1.2 *Test Results*

Test No.	**Expected Results With Above Inputs**	Pass?
1	Successfully opened window showing output from webcam.	✓
2	Successfully opened window showing output from webcam.	✓
3	Successfully opened window showing output from specified video.	✓
4	Successfully opened window showing output from specified video.	✓
5	Error 'AssertionError: Must supply a video if not using webcam'	✓
6	Error 'AssertionError: Must supply a video if not using webcam'	✓

2.1.1.3 *Notes on results*

Everything passed as expected.

2.1.2 *Ball_Tracker.angle(x1, y1, x2, y2)*

The function angle(x1, y1, x2, y2) is used to compute the angle between two given points. This is used to calculate both the exit angle (the angle of the ball as it leaves the player's hand) as well as the entry angle (the angle of the ball as it starts to descend and enter the basket). The method will be called as follows:

```
angle(x1,y1,x2,y2)
```

2.1.2.1 *Test Inputs*

Test No.	**Input Params**	**Expected Result**
1	x1 = 10, y1 = 100, x2 = 10, y2 = 200	90
2	x1 = 25, y1 = 90, x2 = 200, y2 = 500	67
3	x1 = 20, y1 = 200, x2 = 150, y2 = 600	72
4	x1 = 0, y1 = 350, x2 = 100, y2 = 400	27
5	x1 = 0, y1 = 50, x2 = 300, y2 = 550	59

2.1.2.2 *Test Results*

Test No.	**Actual Result**	**Pass**
1	90	✓
2	67	✓
3	72	✓
4	27	✓
5	59	✓

2.1.2.3 *Notes on results*

The angle function was able to pass all test cases presented. It is worth mentioning that floating points were not accounted for. Instead the function simply converts and rounds the answer to the nearest integer value as more precision is currently unnecessary.

2.1.3 *Ball_Tracker.direction(pts, direction, frames)*

The function direction(pts, direction, frames), where pts is a large list of points (x,y coordinates) corresponding to the location of the ball on screen, the direction is the direction you are checking and the frames is the number of frames you are checking it for. Note the directions are represented by 4 numbers (0,1,2,3) which correspond to up,down,left and right respectively. This function is used to confirm if the ball is going in a certain direction before beginning the calculation of its angle. For instance, if the descending angle of the ball is desired and the player is facing left, the program will use this function to detect when the ball starts moving in the down left direction.
Method will be called as follows:

```
Ball_Tracker(is_live, video_path).direction(pts, direction, frames)
```

2.1.3.1 *Test Inputs*

Test No.	Input Params
1	*pts* = [(0,2),(0,4)], *direction* = 0, *frames* = 3
2	*pts* = [(0,4),(0,2)], *direction* = 0, *frames* = 3
3	*pts* = [(2,0),(4,0)], *direction* = 1, *frames* = 3
4	*pts* = [(4,0),(2,0)], *direction* = 1, *frames* = 3
5	*pts* = [(2,0),(4,0)], *direction* = 2, *frames* = 3
6	*pts* = [(4,0),(2,0)], *direction* = 2, *frames* = 3
7	*pts* = [(2,0),(4,0)], *direction* = 3, *frames* = 3
8	*pts* = [(4,0),(2,0)], *direction* = 3, *frames* = 3

2.1.3.2 *Test Results*

Test No.	Expected Results With Above Inputs	Pass?
1	Returns True since 2 less than 4	✓
2	Returns False since 4 not less than 2	✓
3	Returns True since 4 greater than 2	✓
4	Returns False since 2 is not less than 4	✓
5	Returns True since 2 less than 4	✓
6	Returns False since 4 not less than 2	✓
7	Returns True since 4 greater than 2	✓
8	Returns False since 2 is not less than 4	✓

2.1.3.3 *Notes on results*

The direction function was able to pass all test cases presented. However there might be other cases that are yet to be accounted for. For instance, while it is unrealistic, if the ball were to remain completely still at some point, the coordinates (0,0) would fill up the points list. And the right now it does not account for this edge case so the function can be improved upon.

2.2 NETWORKING MODULE

The following will be manual testing of the post functions in the Networking python module. The Networking_Python class is instantiated as follows:

```
Networking_Python(entry_angle, exit_angle, arc_height, zone = 0)
```

where *entry_angle*, a float, is the calculated angle at which the ball enters the net, *exit_angle*, a float, is the calculated angle at which the ball leaves the shooter's hands, *arc_height*, a float, is the maximum height of the arc travelled by the ball and *zone*, an integer, is the zone of the court where the user is shooting from.

The Networking_Hardware class is instantiated as follows:

```
Networking_Hardware(time_of_shot)
```

where *time_of_shot* is the time the shot that went in the basket was taken. The Arduino will only send this information to the server if the shot went in.

2.2.1 *Networking_Python.post(url)*

The function post will send the data the specified url that was received from the initialisation of the Networking_Python. The user_id is also a needed parameter of the post(url) method, although it is not in the method signature as it needs to be added at a later time using the *set_user_id(user_id)* method. The method will be called as follows:

```
Networking_Python(entry_angle, exit_angle, arc_height, zone).post(url)
```

2.2.1.1 *Test Inputs*

Note: The url for every test will be **http://54.145.183.186/api/shot**, which is api url for adding a shot to the database.

Test No.	Input Params
1	*user_id* = 1, *entry_angle* = 45.0, *exit_angle* = 45.0, *arc_height* = 3.5, *zone* = 0
2	*user_id* = 1, *entry_angle* = 45.0, *exit_angle* = 45.0, *arc_height* = 3.5
3	*user_id* = None, *entry_angle* = 45.0, *exit_angle* = 45.0, *arc_height* = 3.5, *zone* = 1
4	*user_id* = 0, *entry_angle* = 37, *exit_angle* = 56, *arc_height* = 5, *zone* = 3
5	*user_id* = 1, *entry_angle* = 43.43, *exit_angle* = None, *arc_height* = 4.27, *zone* = 4
6	*user_id* = 1, *entry_angle* = 42.34545, *exit_angle* = 56.34345, *arc_height* = 3.54, *zone* = 1
7	*user_id* = 1, *entry_angle* = None, *exit_angle* = None, *arc_height* = None, *zone* = None

2.2.1.2 *Test Results*

Test No.	Expected Results With Above Inputs	Pass?
1	The data was successfully added to the server's database.	✓
2	The data was successfully added to the server's database.	✓
3	Error 'AssertionError: Must login before posting to server'	✓
4	Error: Error page returned stating the user with $id = 0$ cannot be found.	✓
5	Error: Error page returned stating the server could not find exit_angle.	✗
6	The data was successfully added to the server's database.	✓
7	Error: Error page returned stating the server could not find entry_angle.	✗

2.2.1.3 *Notes on results*

Test cases 5 and 7 failed the unit test for the same reason: the post(url) method is not enforcing that input parameters cannot be None, nor is the server ensuring that the data it receives is not NULL or empty.

2.2.2 *Networking_Hardware.post(url)*

The function post will send the data the specified url that was received from the initialisation of the Networking_Hardware. The user_id is also a needed parameter of the post(url) method, although it is not in the method signature as it needs to be added at a later time using the *set_user_id(user_id)* method. The method will be called as follows:

```
Networking_Python(time_of_shot).post(url)
```

2.2.2.1 *Test Inputs*

Note: The url for every test will be **http://54.145.183.186/api/shot**, which is api url for adding a shot to the database.

Test No.	Input Params
1	time_of_shot = '2017-03-26 14:44:04'
2	time_of_shot = '2017-26-03 14:44:04'
3	time_of_shot = ''
4	time_of_shot = None

2.2.2.2 *Test Results*

Test No.	Expected Results With Above Inputs	Pass?
1	The data was successfully added to the server's database.	✓
2	Error: Error page returned stating the time_of_shot is an invalid time.	✗
3	Error: Error page returned stating the time_of_shot is an invalid time.	✗
4	Error: Error page returned stating the time_of_shot is an invalid time.	✗

2.2.2.3 *Notes on results*

Test cases 2 - 4 failed due to the fact that the post(url) method is not ensuring that the recieved time is a valid time. A valid time in this case is YYYY-MM-DD HH:mm:ss, and the time cannot be None or an empty string.

3 HARDWARE TESTING

3.1 BASKET TRACKER

The basket tracker portion of the project is made up of three components: the Raspberry Pi computer, the distance sensor and the wifi module. All three of these components are assembled together with a bread board and will be mounted under the net. Ideally, if we had the resources, we would design a chip to fit right into the bread board, but the current setup functions all the same.
The testing for this section will be more informal due to the fact that it's difficult to unit test hardware components.As one will notice, all three components are tightly integrated and cannot accurately be testing alone.

3.2 DISTANCE SENSOR

Testing the distance sensor is making sure it functions the way it is intended to function. The only way to test the distance sensor is to hook it up to the Rasbperry Pi and look at the output. There is a video on our GitHub repository of this being testing and the testing is a success. In the video, one will note the numbers on the computer monitor. These numbers are the distance between the range detectors and the basketball. When the basketball is "shot", the numbers temporarily stop, signifying the ball has gone through the net. The name of this video is *Range_Detector_Test.mp4*.

3.3 WIFI MODULE

Testing the Wifi module is similar to testing the distance sensors; it can only be testing through the Raspberry Pi computer. The WiFi module gives the Raspberry Pi the ability to connect to local Wireless Hot-spots and connect to the Internet so as to create a post request with the data gathered from the Distance sensors. This has also been tested and is working. The proof of functionality is the test report on the Network_Hardware module as seen in Section 2.2.2. This test requires the Raspberry Pi to have Internet access.

3.4 RASPBERRY PI

Testing the Raspberry Pi is ensuring that the computer itself actually functions, and that we can get the distance sensors and the WiFi module functioning through the Raspberry Pi computer. The success of this test is seen in the fact that the tests for the distance sensors and the WiFi module were a success. This means that the Raspberry Pi is doing its job of computing whether the ball made it into the basket and sending that data to the Laravel server.

4 TRACEABILITY

The following section is to display the relationship between the test cases and the requirements for this project. With this the design decisions made for this project will be justified.

4.1 REQUIREMENTS

Test	Description	Requirement
2.1.1	Confirms video/webcam has been connected	6. Record Data
2.1.3	Confirms proper direction is being verified.	7. Give Feedback
2.1.2	Confirms proper angle is being computed	7. Give Feedback
2.2.1	Confirms shot data can be sent to the server	5. Upload Data
2.2.2	Confirms shot data can be sent to the server	5. Upload Data

4.2 MODULES

This section lists the specific modules being tested.

Module name	Test Type	Requirement
Ball_Tracker(is_live, video_path).begin_capture()	Unit test	6. Record Data and 7. Give Feedback
Ball_Tracker.direction (pts,direction,frames)	Unit test	6. Record Data and 7. Give Feedback
Ball_Tracker.angle (x1,y1,x2,y2)	Unit test	6. Record Data and 7. Give Feedback
Networking_Python.post (url)	Unit test	1. Encrypt Data, 2. Decrypt Data, 4. Authenticate User, 8. Access Data and 9. Retrieve Data
Networking_Hardware.post (url)	Unit test	3. Store Results, 5. Upload Data, 6. Record Data, 7. Give Feedback and 10. Hold Connection

5

CHANGES AFTER TESTING

Performing the unit testing and hardware testing illuminated some bugs that need fixing before the final demonstration. Some of the bugs that were caught during the unit testing were due to verification; we need to make sure that the data coming to both the Ball_Tracker module, the Networking module and the Laravel server is valid data. There were also edge cases that were not considered such as as if the location of the ball does not change. These issues were realised thanks to the extensive test cases used and will be corrected for the future.

The fixes for all of these bugs are simple yet time consuming. We need to at every stage make sure that the incoming data is valid. This will be done using if statements to not only check if a variable is None, but also to make sure that the Networking module is not sending a floating point number when the server needs an integer. Exception handling may also be used to deal with possible unexpected user input that would cause errors in the application.

As for the hardware, all of the tests were a success. That being stated, changes to the hardware are still foreseeable - potential changes include moving the hardware to a circuit board. Also the hardware currently may prove challenging for the average user to setup. If it is not setup correctly, this will also result in unforeseeable errors during run-time. These issues will have to be resolved as the project moves forward.