FollowThrough: Design Document - Revision 0

Djordje Petrovec
1306258
petrod@mcmaster.ca

George Plukov
1316246
plukovga@mcmaster.ca

Zayan Imtiaz
1152665
imtiaz3@mcmaster.ca

Philip Habib
1310455
habibp@mcmaster.ca

January 6, 2017

CONTENTS

0.0.1 *Revision Chart*

Table 1: Revision Chart

Revision Number	Revision Date	Description
1	January 6th 2017	Starting the document.
2	January 10th 2017	Updating large amounts of components.
3	January 11th 2017	Finishing touches for revision 0.

1

SYSTEM ARCHITECTURE

1.1 INTRODUCTION

This document details the general system architecture design of FollowThrough. It goes in to detail on the major decisions that were made at key points throughout the product. It also discusses the changes that are likely and unlikely. Lastly it mentions how the requirements are linked to the design.

1.1.1 *Terms Used*

Table 2: Terms Used

Term	Description
FollowThrough	The name of the product.
Arduino	The processor for the hardware used.
Form	Refers to the user's shot form in Basketball.

1.1.2 *Tables Used*

Table 3: List of Tables

Number	Name
2	Terms Used
3	List of Tables
4	List of Figures
5	Pro/Con Chart
6	Traceability Chart

1.1.3 *Figures Used*

Table 4: List of Figures

Number	Name
1	Uses Hierarchy

1.2 OVERVIEW

1.2.1 *Purpose of the Project*

This product is to be used by the common Basketball player so that they may improve their form without having to go through the expenses of a personal coach. FollowThrough is an inexpensive tool which provides the user real time feedback while they are on a Basketball court.

1.2.2 *General Description*

As mentioned previously this product gives the user real time feedback so that they may improve their form. The data gathered on site is recorded and then sent to an online database. The user may afterwards review all the gathered data through an online interface. This will maximize the potential for improving the user's form.

1.2.3 *Document Structure*

This document was separated into two sections; the first being System Architecture and the other being Component Design.

1.3 IMPORTANT DESIGN DECISIONS

1.3.1 *The Online Interface*

This portion of the product was included because we realized that a user would very likely want to review all their data on another date. While receiving real time feedback is the main point of the product, in some cases the information can be overwhelming. This is where the online interface comes in to play. On top of that there would be no reason to memorize what needs to be changed in the user's form, because it's all online. This only takes work off of the user and makes the product more convenient.

1.3.2 *Databases*

The tables that the server will hold will be as follows: A user table and a shot table. The user table contains a user ID (Primary Key), a user name (Unique), an email (Unique), a password, a first name and a last name. The shot table will contain a shot ID, date

the shot was made, user ID (Foreign Key) and data on each shot such as location it was taken, distance from the net, over or under shot, details of the arc and if the shot was made or missed. The tables will have a one to many relationship from users to shots. This will ensure that the system is scalable and none of the information is missed.

1.3.3 *Haar Cascade Files*

A major issue addressed at the beginning of this project is having an effective way to track the ball. We decided to use Haar cascade files to solve this issue. A Haar cascade file is a file which compiles all the features of an object by putting together many images of that object. This accounts for color, texture, lighting, shape... etc. The cascade file will take a long time to compile, so it will be compiled ahead of time, prior to launch. The other options were to simply track by color, shape or size. All the options were outlined below in the Pro/Con chart.

Table 5: Pro/Con Chart

Method	Pros	Cons
Size Tracking	Simple to implement, fast to run.	Cannot account for depth.
Color Tracking	Simple to implement, fast to run.	Cannot account for background objects.
Shape Tracking	Simple to implement, fast to run.	Cannot account for depth.
Haar Cascade	Removes all noise.	Takes a long time to create.

Regardless of the fact that all other options were simpler to implement and quite fast to run on hardware, background noise was a big problem with them all. Haar cascade files were chosen because it removes noise, and the only con is remediable by pre-compiling the files, as it only needs to be compiled once.

1.4 LIKELY AND UNLIKELY CHANGES

The following two subsections will cover what is likely to change and what is unlikely to change in respect to design elements.

1.4.1 *Likely Changes*

- The GUI of the software will be made friendlier.
- The tracking will expand to different colors of Basketballs.
- Changing the current implementation to work with more hardware. This could include tracking the player's position and keeping track of score.

1.4.2 *Unlikely Changes*

- Logging in and out of each system.
- Expanding into other tracking goals.

- The tracking method.

1.5 MODULE HIERARCHY

1.5.1 *Decomposition of Components*

The decomposition of FollowThrough will be minimal. There are two main components: the python module that houses our algorithm for tracking the ball and calculations, and the web application that will receive and display data from the python module. These components are made up of either system libraries or OpenCV libraries.

1.5.2 *Uses Hierarchy*

The following diagram is the uses hierarchy for FollowThrough. Note that system libraries will be included but the modules that the system libraries depend on will not be included.

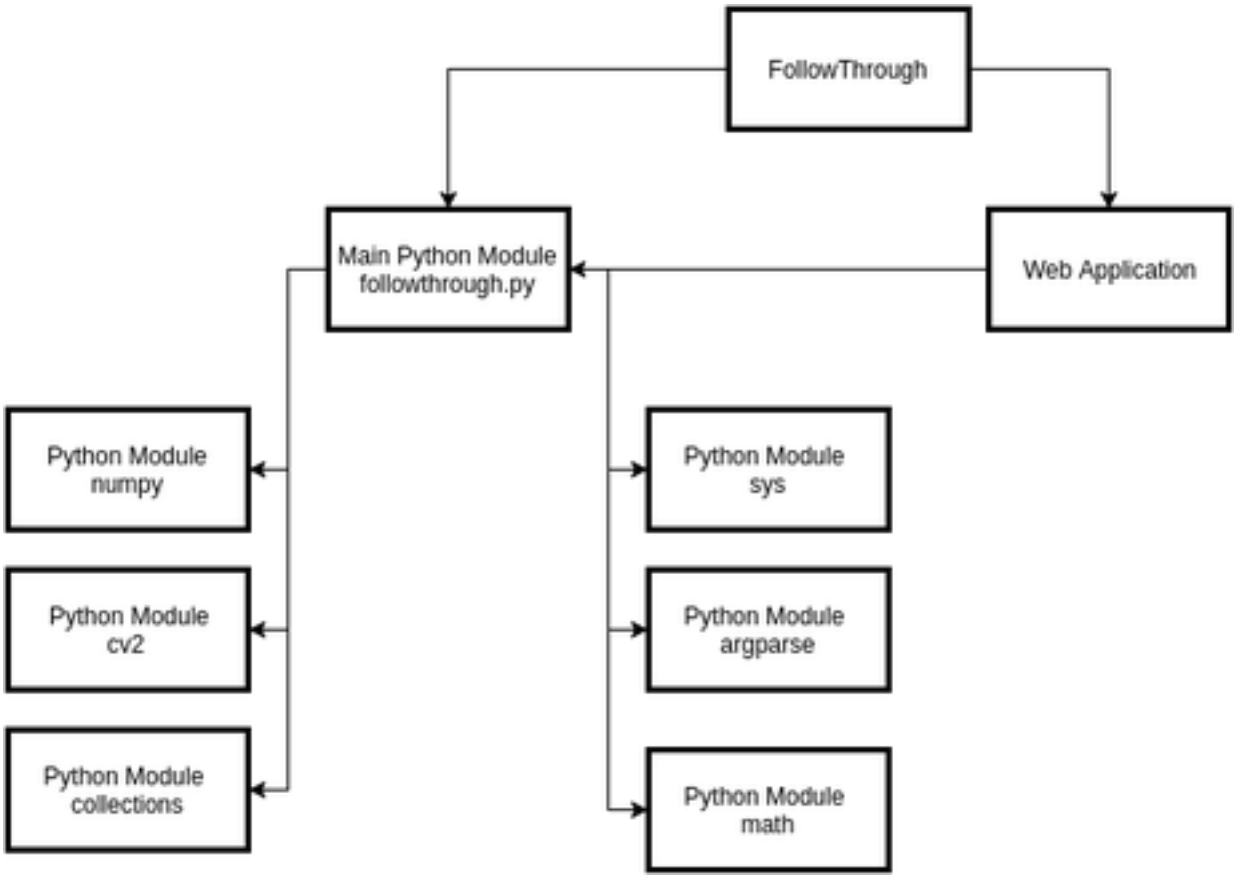

Figure 1: Uses Hierarchy

1.6 REQUIREMENTS TRACEABILITY

The chart below explains the connections between the requirements and how they are linked to the functional requirements in the requirements document.

1.6.1 *Traceability Chart*

Table 6: Traceability Chart

Requirement	Link
Upload Data	This is necessary to obtain the online interface.
Give Feedback	This is needed for the user to receive real time feedback.
Access Data	This is also needed for the online interface portion of the product.
Retrieve Data	This is crucial for the online interface.
Record Data	This is also crucial for the online interface.
Store Results	This is paired with the Record Data requirement.

2

COMPONENT DESIGN

2.1 INTRODUCTION

2.1.1 *Terms Used*

Table 7: Terms Used

Term	Description
FollowThrough	The name of the product.
Arduino	The processor for the hardware used.
Form	Refers to the user's shot form in Basketball.

2.1.2 *Tables Used*

Table 8: List of Tables

Number	Name
7	Terms Used
8	List of Tables
9	API Route
10	Angle Function
11	Haar Cascades
12	Mean Shift

2.1.3 *Technologies and Languages*

This product is developed using OpenCV for all of it's tracking. The language used for OpenCV is Python. Most of the development will be done here. The language for the online interface has not been chosen yet but it would likely be HTML/JavaScript/PHP.

The hardware for this product will mostly be on the user's laptop. This is mostly for the tracking, as to reduce the costs for the product the user only needs to use a simple web cam. Any additional hardware, such as for tracking score, will be done with an Arduino.

2.2 API

2.2.1 *Overview*

The web API provides the hardware and openCV applications the ability to store user data and track statistics over a long term. This provides a line of communication for the three separate applications and the SQL database which stores all user data. The API also provides all the user data for the front end of the web application.

2.2.2 *API Outline*

Table 9: API Route

URL	Method	Purpose
/api/shot/{user.id}	PUT	Insert shot info provided by python into database for user with specified $user_id$.
/api/make/{user.id}	UPDATE	Update shotMade field for last shot taken by user with id, $user_id$.
/api/Users/	GET, PUT, DEL	Add, edit or retrieve information about a user.
/api/Users/shots/	GET	Retrieve a JSON object containing all user shots taken (made and missed)
/api/Users/shots/{var}/	GET	Retrieve a JSON object containing all user shots taken (made and missed) based on a parameter

2.3 KEY ALGORITHMS

OpenCV has countless built in algorithms to perform various tasks. Aside from those we implemented an angle algorithm. We also plan on implementing Haar Cascades as well as a Mean Shift Algorithm to further improve the program's ball tracking capabilities. As of right now, the program only uses colour detection for tracking the ball. This has issues such as other similar colours and lighting interfering with the computations so these two algorithms when implemented will rectify these issues.

2.4 ALGORITHM 1 - ANGLE CALCULATION

Given two points, it computes the angle between them and returns it. This is utilized after the ball reaches maximum height. This is because we want to compute the angle from the maximum height to when it starts descending as well as finally landing in or near the basket. There is an ideal angle contingent on a person's height the shot should form before hitting the basket and thus it is important that our program calculates this correctly and efficiently in order to give the user sufficient feedback.

2.4.1 *MIS Chart - Angle Calculation*

Table 10: Angle Function

Function Name	Inputs	Output
angle	coordinates of two points: x1,y1,x2,y2	Angle(in degrees)

2.4.2 *Likely Changes*

The angle computed right now does not account for the height of the user. Research indicates the ideal angle will vary by height so the algorithm should also take height into account in the future.

2.5 ALGORITHM 2 - HAAR CASCADES

This is a machine learning algorithm we plan to implement in our program in order to improve object tracking. In basic terms the algorithm requires a large amount of positive images (images with a basketball) and negative images (images without a basketball) to analyze. From these images it extracts features from the positive images and tries to detect the distinct the features of the positive image, in our case a basketball being present. These features can include edges, lines and shapes. Note that because this is a machine learning algorithm, it would only have to be run once. Once it obtains sufficient data, this state will be stored and it will be simply implemented alongside the currently working program.

2.5.1 *MIS Chart - Haar Cascades*

Table 11: Haar Cascades

Function Name	Inputs	Output
haarCascade	array of positive images, array of negative images	State file which detects positive images

2.5.2 *Likely Changes*

Haar Cascades look for all kinds of features to distinguish images but in reality, only a few features will likely be needed as a basketball is a relatively simple object. Regardless of that fact it does account for lighting and a few other features that other algorithms can't. For this reason we will likely be putting this method into motion, but the changes that can be done to Haar cascade files is naturally limited.

2.6 ALGORITHM 3 - MEAN SHIFT ALGORITHM

This algorithm given a specific window size simply detects the part of an image with the most amount of pixel density within that given window size. For instance if the image is a blank canvas with scattered dots and a circular window is specified, it will attempt to move that circle to the object with the highest concentration of dots. The plan is to implement this algorithm so our program is not only detecting color but also the pixel density of a basketball. Normally the program relies on no other colour being present in the scene that is too similar to the basketball colour. But once this is implemented, even if there is another object on screen with the exact same colour values, it would also need a similar pixel density to interfere with the program. Hence, this greatly reduces the chances of interference from surroundings during image recognition since there are two checks occurring, not just one.

2.6.1 *MIS Chart - Mean Shift*

Table 12: Mean Shift

Function Name	Inputs	Output
meanShift	specified detection window size and shape	object with highest pixel density (basketball)

2.6.2 *Likely Changes*

It is still unclear if the basketball will have the highest pixel density on screen. It is possible this is not true in which case the algorithm will have to be slightly modified to detect a known range of pixel densities that basketballs generally have.

2.7 ADDITIONAL COMMENTS