Final Project Report: A Face Detection Controlled Game

Andrew Brusso

April 2020

Abstract

I built a simple game using the Unity engine. The game centers on a ball rolling down a board, with various obstacles and platforms on it, similar to Pinball or Pachinko. The game uses a web camera as the primary input device, by leveraging OpenCV and Deep Learning to detect the player's face, and convert the player's movements into movement in the game.

Project Description

My project involved building a game in Unity. The gameplay focuses on a ball that is rolling down a vertical board, with different platforms and obstacles in the way. The goal of the game is to get from the top of the board to bottom, after which the level is completed. The game itself isn't very inspired, but the code happening behind the scenes to control it is pretty interesting.

The game consists of three main components: the game environment, face detection and the camera server, and the client/server interface that connects them together. The game environment is the Unity code that keeps track of the game state, including the direction of the camera and movement, whether the game is paused, the game menus, and other details of the game itself. The face detection and camera server is a threaded application written in Python, which requests frames from a connected USB camera (the camera model I am using is the Logitech C920). The detection pipeline performs processing on the camera's video feed, and then converts it into information that can be used by the game as input (in particular, the orientation of the players head). The client/server interface is a pair of network sockets that facilitates the communication between the game environment and the camera server, allowing the game environment to connect to the camera server and request the orientation of the players head at frequent intervals.

When the game is started, the camera server is automatically started as well, and the game client repeatedly tries to connect to the camera server. If a camera server cannot be connected to (because there is no camera connected to the computer), the game then displays a warning message, and allows the player to continue using keyboard based controls instead. The camera server relays information to the game client, which rotates the gameboard based on the orientation of the player's head, which then changes the direction that the ball moves in.

Implementation

To implement this project, I developed and tested each of the three components in partial isolation to demonstrate that I would be able to implement them individually. Once implemented, I began connecting the components together, and building up sensible logic for how they would interact with each other.

Face Detection and the Camera Server

The first thing I worked on with the project was trying to get some very basic facial detection code running, to prove to myself I would be able to get *something* working. The first step towards this was a very brief tutorial that I followed that implemented detection using the built in Haar Cascade classifiers in OpenCV[4]. From here I started reading through the official documentation for the python implementation of OpenCV [2], to learn what the broader capabilities of the library were. The classifiers in OpenCV were a good starting point, but after some research I found that the library Dlib offered a more comprehensive approach that would allow for detection of specific features by utilizing a much more robust deep learning model (I was very impressed with its detection accuracy) [5].

With Dlib detection in place, my next step was to get this to control something, just to prove I could extract some information from this detection. To do this, I captured the eye positions from the detection model, and wrote some code to calculate the angle from a line drawn between the eyes and the horizontal axis (using the eyes as a sort of "level", see Figure 1). I used a python library that I have some previous experience with called pyautogui to move the mouse cursor up and down based on face orientation: moving up when my head was tilting one way, and down when tilted the other way. This was working, but there was a notable amount of latency, so I started researching potential ways to make my code more efficient.

One of the easiest optimization was simply to drop the resolution from the camera feed, which helped a lot (this trades off efficiency for accuracy, but Dlib still performs well even at low resolution). Next, I found some articles on approaches to thread out some of the I/O operations involved in reading from the camera to reduce the amount of time spent waiting [6]. One final thing I started noticing was that when I was sitting still there was a small amount of noise causing the input to jitter around. This happened to be at the same time we started talking about filtering in class, so I implemented a basic averaging filter, which seemed to work well enough. This left me fairly close to the final

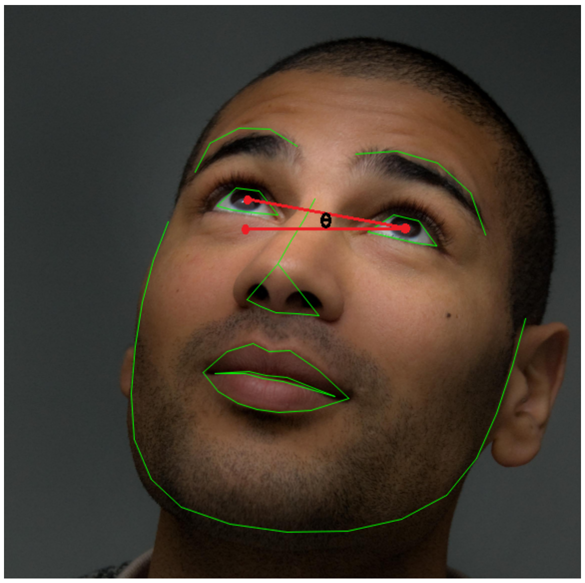


Figure 1: The green lines are depictions of landmarks being detected by the Dlib python library. Landmarks include the eyes, eyebrows, mouth, nose, and outline of the face. The angle θ is the angle that my code calculates, to determine the orientation of a person's face. [3]

pipeline that I'm using for the project now. The rewrite to make things threaded also gave me an opportunity to convert my code to being more object oriented and modular, so that I now have the option of expanding to additional input options, or different detection techniques more easily.

Game Environment

Part of the way through building the facial detection component, I began researching a lot of introductory material for Unity, including starting to read through the Unity user manual [9], and watching a variety of basic video tutorials on Youtube. My first game that I created was by following the official Unity tutorials for a ball rolling game [8], which gave me a basic understanding for some of the concepts in Unity. From there, I had a basic game environment working, which made me comfortable enough to switch back to working on the detection, and how to integrate it into the game environment. This is when I built out the client/server interface, and started integrating it into Unity (which I discuss in the next section). When I was able to successfully start reading the camera inputs into Unity, I started switching back to focusing on the Unity aspects of the project more.

This is when I followed a very useful series of videos by a Youtuber named Brackeys [1]. Following this series is probably the first time I started actually feeling like I was making a game, since these tutorials helped me build out a menu, end game conditions, and a way to transition between different levels in the game using Unity's scene manager. At the time of writing, I only have a single level, which is a very basic one I was using mostly for testing, but I have things setup so that adding and designing additional levels should now be an easy task. I have been working on polishing up some of these game aspects to make it feel like the game actually has some mechanics and thought to it (although that never was the primary focus).

Client/Server Interface

Once I had a basic game environment setup, I started thinking about how I was going to get information from the camera into Unity. I looked into a lot of potential approaches for this, but I found that the most promising one was simply using a client/server paradigm, by having Unity connect to a camera server to request information on what the state of the camera currently is. My starting point for this was a github repository [7], which utilized the networking library ZeroMQ. I cloned this repo and starting reading through the code to see how it worked. I was thinking of simply rewriting it completely to just be basic $C\#$ and Python sockets, but I found I liked the simplicity of ZeroMQ, so I elected to use it in my implementation as well. Initially, I was separately running a client and server in two separate processes to confirm I could send and receive data to and from Unity. This was promising, but I found that there was a lot of issues with the approach as is, because when the game or server was stopped, or the connection lost, or the webcam was unplugged, then I would have to restart both the client and the server.

That is when I started working on getting a client that would run persistently whenever the game was running in Unity, and would poll and reconnect both on the client and server side. This helped with the issues of reconnecting, but I was still separately starting up a server manually in python and then the client in Unity. This was when I went down the rabbit hole of trying to run Python code from within Unity.

My first attempt was using IronPython (a $C\#$ implementation of the Python interpreter), because I saw a lot of forum posts casually indicating that this was the proper way to use Python in Unity. After hours of searching, I finally was able to identify all of the DLLs I needed to include to get it working properly. I then tried to pip install the Python libraries I needed, and lo and behold I ran into a handful of Python errors that lead me to the conclusion that OpenCV and Dlib simply weren't going to be compatible with IronPython. I then decided to just create a regular Process in $C\#$, in what amounts to an exec call. This worked out great, and I was now able to simply press the play button in the Unity developer studio, and have the camera running automatically. This solution relies on python being installed with the correct libraries, and being on the windows PATH, so while it works well, it wouldn't be able to be used in a properly delivered game.

When I started working on the functionality for multiple levels and scenes, I also did some work to convert the camera client in Unity to being a singleton class, and prevented it from being destroyed at the end of each level. This made it so that the camera connects automatically while in the main menu, and shouldn't ever have to reconnect. As a final finishing touch, I added detection for whether the camera was connected or not in game, so that if you plug in or unplug the camera while playing it will pause the game and give you the option of choosing to use the webcam or keyboard controls. I now had a full camera processing pipeline, that runs automatically when the game is started inside of Unity, and is able to communicate information to Unity to control elements of the game. With that I returned to building and improving the Game Environment, which is where I currently am at.

Did I succeed?

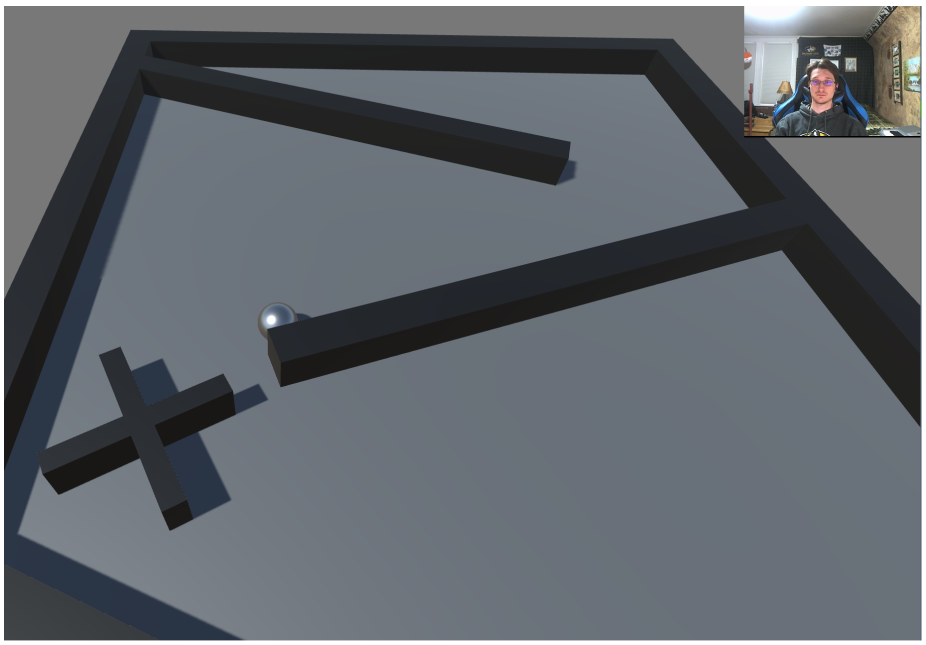
The question of whether I succeeded in completing what I intended to do is probably best answered by referring back to the goals I laid out in my proposal:

1. Create a facial detection pipeline in OpenCV
2. Control something basic via the face detection
3. Build a basic environment in Unity
4. Create an interface/plugin for OpenCV to communicate with Unity
5. Improve the face detection, and explore its limits
6. Improve Unity environment into something "game like"

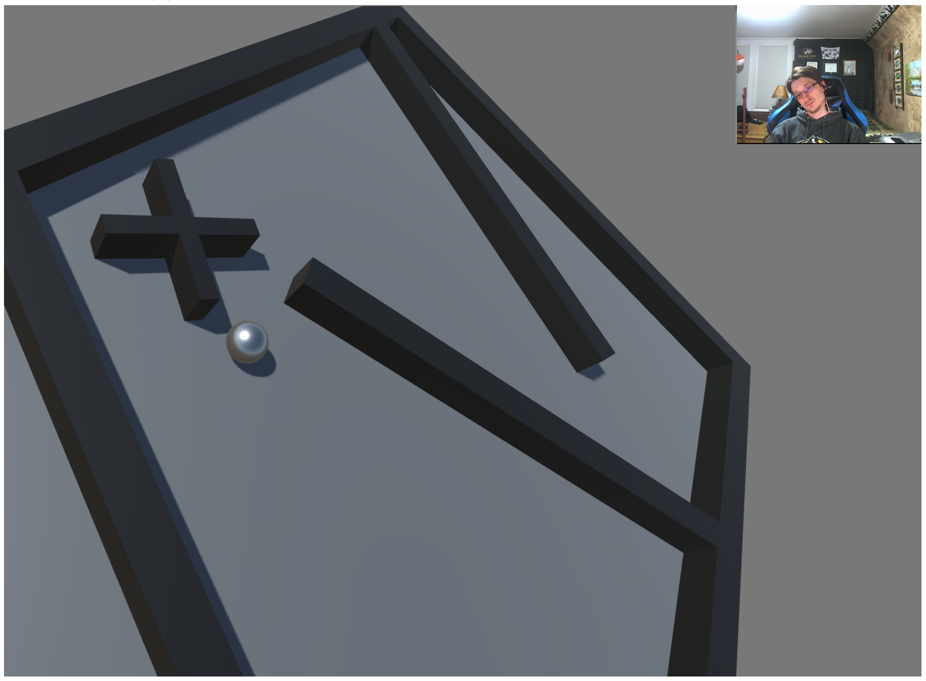
Looking at this list, I would say that up to goal number 5 I met all of the goals without a doubt. For goal 5, I would say I indirectly explored some of the limitations, such as how the face detection performs from different angles and perspectives, and with different faces entirely. That said, I think there is undoubtedly still room for improvement (and likely always will be). For goal number 6, I would argue that the current state of the game is approaching being game like now (see Figure 2 for the level I made for testing face detection), and I think that all the other parts are established enough that this could now be a focal point. Honestly though, I don't think I could currently claim that the end product, while now resembling and having many of the necessary elements of a game, is something that I would intentionally seek out and play.

Another potential measure of success is that I did have a few people try the game out. Reactions were generally positive with what I have so far, but it was clear that they were underwhelmed with the actual game portion, and more so impressed with how it was working (which makes sense, because that is the bulk of the project).

To then answer whether I achieved what I set out to do, I think I did succeed in my project. At a glance, the game itself isn't incredibly impressive, but I'm pretty happy with the amount of work I put in, and I think I learned a lot



(a) The gameboard when a player is sitting with their head level



(b) A tilted game board resulting from a player tilting their head

Figure 2: Depiction of the first (and currently, only) level in the game. a and b show the change that occurs as a result of tilting your head in game, and both show the obstacles and game elements currently in place.

about OpenCV and Unity along the way. I would definitely recommend for other student's to try something similar, since this proved to be a very easily scalable project. While I sort of dived into the deep end for the project, using a bunch of technology I've never used before, I think it lead to me learning a lot about both OpenCV and Unity. Having worked with both of these technologies now, I definitely see myself using OpenCV for some personal projects in the future.

When I started this project, I wanted to see if I could identify why webcams aren't currently a broadly used input device for games. At the end now, I think I can conclude that the issue isn't of technology, but due to the limited scope that is created by using only a persons face as an input device, and possibly also due to market pressures or consumer attitudes. I'm not sure that I'll ever go much further into the world of game development, but I certainly value having some insight into what developing modern games looks like, and I think trying to do so at least once is a valuable experience to have.

References

[1] Brackeys. *How to make a Video Game - Getting Started*. Youtube. 2017. URL: `https://www.youtube.com/watch?v=j48LtUkZRjU`.

[2] Alexander Mordvintsev Abid K. *OpenCV-Python Tutorials*. 2013. URL: `https://opencv-python-tutroals.readthedocs.io/en/latest/index.html`.

[3] Davis King. *Real-Time Face Pose Estimation*. Dlib. 2014. URL: `http://blog.dlib.net/2014/08/real-time-face-pose-estimation.html`.

[4] Adarsh Menon. *Face Detection in 2 Minutes using OpenCV Python*. Towards Data Science. 2019. URL: `https://towardsdatascience.com/face-detection-in-2-minutes-using-opencv-python-90f89d7c0f81`.

[5] Adrian Rosebrock. *Facial landmarks with dlib, OpenCV, and Python*. pyimagesearch. 2017. URL: `https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/`.

[6] Adrian Rosebrock. *Increasing webcam FPS with Python and OpenCV*. pyimagesearch. 2015. URL: `https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/`.

[7] Chanchana Sornsoontorn. *Unity3D Python Communication*. Github. URL: `https://github.com/off99555/Unity3D-Python-Communication`.

[8] Unity. *Unity 5 - Roll a Ball Game - Official Tutorials*. Youtube. 2015. URL: `https://www.youtube.com/watch?v=RFlh8pTf4DU`.

[9] *Unity User Manual*. Unity Technologies. URL: `https://docs.unity3d.com/Manual/index.html`.