Motion Capture Sports Tutor

Final Report

Jonathan Fermin
Jonathan Meade
Stacia Near
Christoph Uhl
Michael Vienneau

CSCI-4830 IML4HCI
Prof. Ben Shapiro
Instr. Will Temple

Table of Contents

Section I: Goals

Mission Statement

We will create a virtual personal trainer that observes a user's performance of correct motions under a coach's supervision, and helps them replicate those techniques later.

Many people are interested in improving their performance at sports techniques, everything from basketball, to boxing, to yoga. Usually this requires expensive, dedicated sessions with a personal trainer, however with our method, the price for improving sports techniques will be more affordable. We seek to develop a method that takes advantage of the skills and insights of professional coaches, but delivers this service directly to your home studio at a low cost.

Using an inexpensive motion capture studio setup, users can compare their motions to the correct form. The idea is to set up the motion capture sensor (Microsoft Kinect) while a coach is present, and have the Kinect extract skeletal data to observe the user's correct form under a coach's supervision. The Kinect input is used to train a computer system using dynamic time warping and classification, to build a database of correct movements. This is an excellent application for machine learning, as gesture detection is a complex problem that is difficult to solve when programming by hand. However, by using machine learning, we can quickly train a system to recognize a wealth of different sports movements. DTW is an ideal choice for this application because it maps realtime gestures dynamically to gestures recorded in training, allowing movements of different speeds to still be categorized correctly.

Another advantage of our system is its ability to recognize how closely a given movement matches the target movement. By default, DTW returns a percentage reflecting how closely a movement matches the target. This percentage is displayed to the user for each location on their body to let them know what part of the motion went wrong. For the purposes of demonstration, the final version of our project focuses only on the basketball free throw. Our team is familiar with this technique and can successfully demonstrate it. However, the goal with the project is to create a framework that could ultimately be used to practice and refine any sports movement.

The system will be a huge benefit for consumers, who can do the initial training while a coach is present, perhaps during a group lesson. Having the coach provide individual lessons is too expensive, so our tutor will provide follow up training at home in the coach's stead. Because machine learning provides a high level of flexibility in terms of allowing the user to retrain the model at their leisure, the user can continue to improve the model that the system considers "correct" each time they visit the coach. This way, the user will be able to see how they consistently improve, and each time their technique gets better, they focus on maintaining that improvement until they have a chance to further refine it with the coach.

Overall, this system will be a huge benefit to hobbyists who don't have the resources to work with a personal trainer regularly, and professionals who want to practice daily at home while having a good standard to measure themselves against.

Technologies

Hardware

- Microsoft Kinect
- 5 Microbits (4 senders, 1 radio receiver)
- Laptop with high processing speed

Open Source Software

- Python base code
- Associated machine learning libraries (SkLearn, etc.)
- Open Source Python DTW algorithm (documentation in repository)
- Python OSC Packet Handling Code
- Team-developed code
    - Feature extraction and smoothing
    - Data handling and passing
    - Microbit data handling (in Processing)

Closed Source Software

- Microsoft Kinect drivers and software components
- User interface portions made in PyQt and Qt Designer (which are closed source)

Target Audience

The main audience of our project is individuals who want to improve their skills in a particular sport, but do not have the financial resources or the scheduling flexibility to meet with a personal trainer or coach on a regular basis. These individuals may have some experience with the sport already, but it is not required because of the nature of the training mechanism. Due to the user-oriented nature of the setup, our system is applicable to a very wide range of individuals with little to no requirements in regards to athletic ability, age, gender, etc.

Users could range from teenagers eager to improve their free-throw style to senior citizens looking to stay active while using specific motions to improve their physical health. The fact that our system contains no pre-trained information makes it perfect for anybody looking to improve their sports posture. In order to train the model on a specific user, they should go through the motion with a coach/trainer. This will enable the system to custom-tailor a solution to each user individually. This will still be cheaper than regularly meeting with a trainer, as the user can decide when the model needs proper updating with a coach (the user can choose to never retrain the model, or to retrain every day → more customizability).

Challenges

Motion capture requires information greater than the majority of the feature extraction used during our previous coursework. Accelerometers and gyroscopes are generally good for objects with single joints, however motion capture requires more detailed information about 3D space. Therefore, the majority of the challenges revolved around gathering input data. Dealing with unfamiliar technologies, noise, and latency was a major factor. Integrating signals from separate sources, in our case the Kinect and Microbits, and analyzing them together was also difficult..

Latency was occasionally an issue with such a large amount of data streaming from multiple Microbits. To simplify this issue, we had four Microbits transmit signals over radio to a single receiver. The receiver was another Microbit connected to the laptop by USB. This greatly reduced latency, and only caused problems when the sending Microbits were obscured from or too far away from the receiver.

Another challenge was getting the DTW algorithm to recognize an improper motion vs. a correct sports motion. Too much information from too many joints can confound the data, so we customized the code that streamed data from the kinect to simplify this aspect. Instead of sending the location of all joints, we only sent upper body joint locations. This allowed the DTW to function more successfully, because only the relevant data was sent. In a basketball throw, the upper body is most important in the throw, so it was not necessary to track the lower body.

As we anticipated, using multiple cameras as inputs led to too much latency and could not be processed in real time. It was too inefficient, and too difficult for the user to set up. Not only did it require extensive hardware to use camera input, it also required a powerful laptop, and in depth calibration each time the cameras were moved. This was not practical for the assignment and was completely replaced by the Kinect.

The Kinect, however, was less accurate than multiple cameras, even though it was easier to use. We chose to use Microbits affixed to the user's hand and arm to stream accelerometer data as a supplement. The kinect does not track wrist movements well, so this was necessary in order to track the wrist portion of the basketball throw.

One of the more intangible challenges was related to the philosophy behind basketball. Even at the professional level, a consensus will never arise between what the perfect form is. Beyond that, we didn't have professionals to consult anyway. As a result, this project's scope was limited to our casual-level knowledge of basketball throws. A fully successful implementation of the project would require expert knowledge.

Our solutions to these challenges are outlined in more detail in the discussion, as well as other problems that arose in connection to these major challenges.

Section II: Design & Architecture

Final Product

This is an outline of the functionality of the final product. In the conclusions, the working parts of the project are discussed in more detail.

Input Hardware

- Kinect correctly obtains upper body joint information
- Microbits successfully track and send acceleration data from the hand and arms

Supported sports

- Basketball free-throw, our standard for testing, is currently the only focus

User interface

- Functional UI with different modes that is accessible to users, see storyboard below
- User can train with a coach, record their own correct motions, and emulate those motions later

User feedback

- Correctness of an attempted movement is displayed as a percentage in user mode

ML model

- DTW is used to match gestures, and integrates with our UI
- SkLearn is used for some additional integration with Python

Major Changes

During testing of our implementation, we revised some of our initial goals due to practical considerations, as well as hands-on experience with what methods were most successful.

Our greatest change from the initial proposal was the cutting out of the PS3 Eye Cameras as a method of input. We included them in the proposal under the assumption that multiple 2D cameras might provide more reliable input than a single 3D camera - the Kinect.

The PS3 Eye Cameras did provide more reliable motion capture. However, they did so too inefficiently - requiring far too much calibration and processing time to be practical for this particular project. Additionally, the Ipisoft Mocap software was still too expensive under the student discount, which prevented us from using it, and would also be an obstacle to potential users. We would need to use our own software in order to prevent features from being locked behind a paywall.

During testing of the PS3 Eye Cameras in a 3-camera configuration with the Ipisoft software, we confirmed that the Ipisoft software is incredibly effective at performing its target task. It failed because our project was too far outside the scope of what the software was developed for, so the software simply wasn't effective for our means. Ipisoft MoCap Studio is designed so 3D animators can capture the skeletal motions of an actor to be mapped onto a 3D model (for instance, a Pixar-style character or similar). This saves a lot of time for 3D animators even if it takes a while to process the camera inputs, so animators do not need real-time processing speed.

It takes several minutes or more to process input from 3 cameras and turn it into motion capture data, depending on the length of the video. For our application, we require real time feedback. This is not only because we need to critique the user's sports technique in real time, but also because our project needs to be simple to use. The user would have to wait for the cameras to process their movement, and know how to input the data back into our application to receive feedback. Instead of introducing this complexity, we decided to focus fully on the Kinect and simply use real time feedback from a single Kinect, easily integrated into our application, to track the user's skeleton.

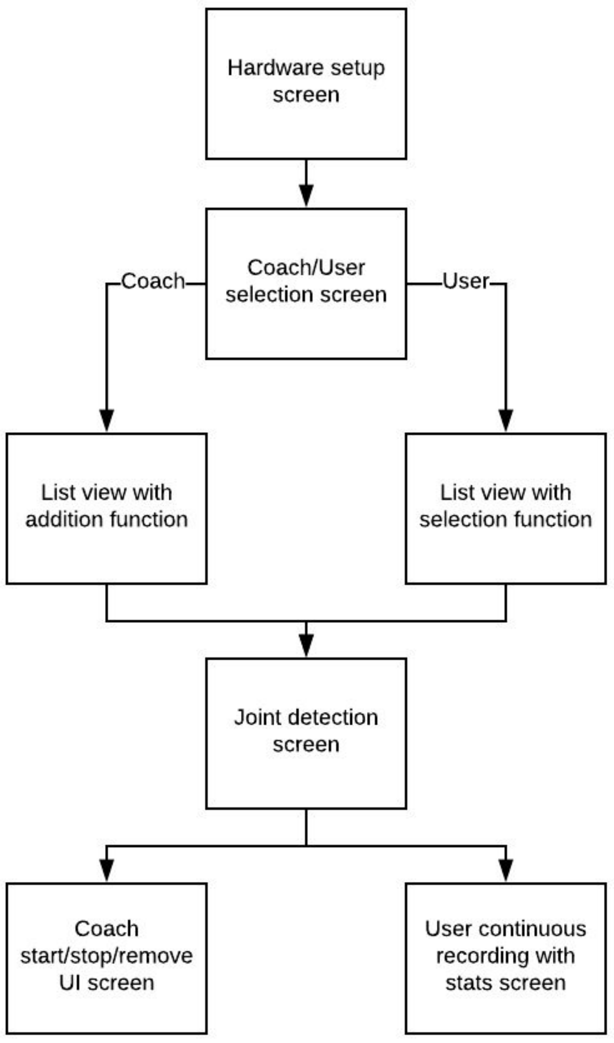
We also did not end up using OpenCV to process visual data in a more customizable way. Time constraints prevented this, along with the phasing out of multiple cameras. We initially planned to integrate with Blender to display the user's movements back to them as an animation. A 3D model of the user's motions would be overlapped with the correct motions to show the user which parts of their body were in the wrong area. Again, this proved unnecessarily complex, and while the feedback from the Kinect is reliable enough to detect major flaws in the performance of a sports movement, it is not reliable enough to create a 3D model of the user and have it move in a convincing or useful way.

Overall, these changes simplified the project and allowed us to focus on the truly important part, which is providing a smooth user experience for clients who want to practice sports movements at home.

Production Timeline

Phases	Tasks	Dates
1 - Concept Phase		
1.1	Decided on our idea	03/05/18
1.2	Wrote up user stories/scenarios with a storyboard and sketches. Finished project proposal.	03/15/18
1.3	In-class status update -- what's happening?	03/19/18
2 - Design & Development Phase		
2.1	Prepare a demo prototype for class - Kinect read user motion well with a black tarp in the background, correctly calculated which motion the user was performing on Wekinator using DTW.	04/02/18
2.2	Figured out how to receive Microbit data and began work on own DTW algorithm..	04/06/18
2.3	Made a UI and finished up DTW algorithm, now only need to connect the two with a pipe and polish up user feedback algorithm and UI.	04/15/18
2.4	Made a pipe to connect frontend with backend, but not quite working yet.	04/19/18
2.5	Worked more on the pipe, still not working..	04/22/18
3 - Final Phase (All major development done, polish over it, etc.)		
3.1	Finally got the pipe to work! Submitted first version of the Final Report, received feedback from Will	04/23/18
3.2	Finished up our Final Report!	04/28/18
3.3	Getting ready for final presentation	04/29/18
3.4	Final Presentation - Prepare our presentation, who will talk about what?	04/30/18

User Interface Flow


Hardware setup screen
Coach/User selection screen
Coach
User
List view with addition function
List view with selection function
Joint detection screen
Coach start/stop/remove UI screen
User continuous recording with stats screen

User Interaction Storyboard

The overall design and mockups of our UI can be found here.

The below storyboard has added "application" dialogue for clarity, but text to speech is not implemented currently in the final version of our product.

Example Coach/User interaction:

User: "Hey coach, can you help me set up the motion capture to help me improve my form while I'm not there?"
Coach: "No problem, what do I do?"
User: "I don't know, let's start up the application.."
[starts up application]
Application: "Thank you for choosing MoCap Trainer! To start, choose either 'Record New Motion' or 'Train a Motion'. "
[screen shows 2 buttons 'Record' and 'Train' -- if no previous recordings, train is greyed out]
[coach selects 'Record']
Coach: "What now?"
Application: "Please add a new motion"
[List of clickable 'plusses' appears to indicate function to add a new motion -- if previous motions have been recorded, they will be listed here]
Coach: "Okay I will add a new motion."
[Clicks a plus, the application asks for a name-input]
Coach: "Let's call it 'Basketball free-throw'"
[Presses enter, UI window for calibration pops up]
Coach: "Okay, stand in the middle in front of the black curtain and spread your arms out."
[User spreads arms out]
Coach: "Hold for 3 seconds."
[Calibration window is replaced by Coach start/stop screen]
Application: "Successfully calibrated! In order to record the perfect motion, the coach needs to press 'start' when the user performs the desired motion, when the motion is over, press 'stop'. Give some buffer time, I will be able to find the motion."
Coach: "Alright, please get into position"
User: "Okay"
[User tries best to perform the motion correctly]

Application: "If the motion was performed well and you think it can be used as training data, please click 'add', otherwise, please click 'repeat'. We will do this 10 times [progress bar pops up], the motions added will work as training data, so please make sure they are correct."

Coach: "Hmmm, I think you need to keep your elbow closer to your body, try again"

User: "Okay"

[Tries again]

Coach: "Perfect, we will use this one! Onto the next, ready?"

. [Continues until done]

.

.

[User by themselves now]

User: "Time to do some training!"

[Starts up application]

Application: "Thank you for choosing MoCap Trainer! To start, choose either 'Record New Motion' or 'Train a Motion'. "

[screen shows 2 buttons 'Record' and 'Train' -- if no previous recordings, train is greyed out]

[User chooses 'Train']

Application: "Choose the motion you'd like to train."

[User chooses 'Basketball free-throw']

Application: "Okay please calibrate"

[User stands with hands spread out]

[Countdown initiates: "3… 2… 1… Click! Calibrated!" is hearable in the background]

Application: "Please yell 'now', to start, and 'stop' to stop. Don't worry about timing, I will be able to recognize your motion"

[Performs motion]

[Percentage of accuracy is displayed on the screen and remains there]

Application: "Good motion, but try to keep your elbow closer to your body!"

[User performs another motion]

[Percentage updates]

User: "Wow! This works so well! Thank you MoCap trainer!"

[Continues to train until the end of days]

Section III: Results

Results and Discussion

As we dove into implementing our theoretical ideas, our methods gave rise to a number of limitations. Our solutions to these limitations are discussed below, along with the implications for the long term success of projects similar to ours.

Hardware - Kinect

Getting the Kinect to reliably send 3D skeletal data was difficult. Though the Kinect is simple to use, its simplicity also comes with some unreliability. The Kinect works by sending out a pattern of infrared dots of known positions. It receives these light dots after they bounce of the environment, become distorted by 3D shapes, and enter the camera again. The infrared light is subject to some of the same issues as visual light. Reflective surfaces can distort the pattern in a way that causes mistakes in how a 3D map is generated. The infrared dots distort in a predictable way when in contact with solid 3D surfaces, but anything that causes the light to bounce unpredictably will interfere with this.

To get around this limitation, we created a solid black backdrop out of diffuse fabric to place behind the Kinect. When the user stands in front of the light-absorbing fabric, they stand out quite well from the environment and are interpreted by the Kinect's 3D sensor with good consistency. The fabric is attached to the whiteboard using magnets, but a modified setup could be used in home as well, perhaps with a PVC pipe frame.

Other problems with the Kinect have more simple solutions. For instance, if the Kinect is placed too far back on the table while looking at the user, the table will take up too much of the Kinect's field of view, even if it is rotated upward. We learned from experience to keep the Kinect rotated upward to capture the high point of a user's basketball throw, and push it as far as possible to the end of the table.

However, if the table with the Kinect on it is too close to the user, this will also cause problems. The user needs to make sure they are far enough from the Kinect that it can capture most of their body within its field of view. The Kinect detects the most likely orientation of each of the user's body parts using the 3D map it detects to construct a skeletal representation. It does this using random forests, an ML strategy that uses the combined output of a multitude of decision trees. Decision trees by themselves are pretty unreliable, but the random forest method is highly effective at classifying information. However, no method is perfect - even this method will fail somewhat frequently on the Kinect, causing the random noise that you see in its skeleton output. Even if everything is set up correctly, the joints on the skeletal representation will jump around statically on screen while the user moves around. If things are not set up correctly, the Kinect cannot perceive your skeleton at all, because there is not enough input for the ML algorithm to make a decision.

This is the reason that the Kinect must be able to see most of your body to work, and if you turn to the side it often loses track of where your joints are. The Kinect is very bad at guessing where your limbs are if anything is obscured behind another body part. As you might guess, this also creates a problem when you cross limbs, making it less clear where the boundary between them is. We worked around all these issues by ensuring the Kinect is far enough away from the user during operation, and only simple sports movements are used. We chose a basketball throw as our base testing algorithm, because the upper limbs do not have to cross to perform this throw.

Note that this is one downside of our implementation that probably cannot be resolved with our chosen input devices - you cannot practice sports movements that require a lot of crossing motions with your limbs. There is no way for us to refine the algorithm to be able to detect crossing of the arms, because the Kinect's default machine learning algorithms are closed source. We know generally how it operates, but Microsoft did not release all information on the technology. As a result, we must choose to keep the movements we track simple for our implementation. Possible future work could use the open source library OpenNI for skeletal tracking instead of the Kinect's default ML algorithm. OpenNI takes in a point cloud (the 3D map that the Kinect generates), and finds the limbs using its own algorithm. With sufficient time, this algorithm could be modified and improved. Alternatively, a combination of two Kinects working together with a software algorithm that averages information between them could be used to provide more accurate results. We did not have access to a second Kinect, so this was not feasible for us.

Once we got the Kinect working, we realized that we still had too many limitations to how we were able to capture data. With just the Kinect, it's very hard to reliably detect the motions of the wrists, and how the wrist is moving relative to the arm. The Kinect doesn't emphasize the detection of where the hand is, so accurate information for this is lacking. We chose to solve this by adding another input device - the Microbit.

Hardware - Microbit

We used a setup in which a single Microbit is hooked up to a laptop using USB. That Microbit serves as the receiver of radio signals from four other Microbits transmitting their location. These four chips are attached to each of the users arms, one on the hand and one on the wrist.

We occasionally ran into transmission issues with four Microbits all sending radio signals to one host. Though this is a lot of information to stream, the receiver seemed to be able to keep up with it. Problems with latency only arose when the signal wouldn't transmit because the sending Microbit was too far from the receiver. We learned to make sure that the receiver was close enough to the user to send a reliable signal.

Attaching the Microbits to the user also proved to be a bit difficult. We want to provide a seamless user experience, in which wearing a device that is required for use of our application is not too burdensome. The device should be comfortable and light. Initially we were attaching the Microbits using tape or rubber bands, which was simply a makeshift solution that resulted in complaints during use testing in the classroom. We are still in the process of acquiring a glove or ipod/phone holding case that we can use to comfortably affix the Microbits to the user's hand and wrist. A fully customized glove would be best, in which the Microbits are affixed sturdily but comfortably to the back of the hand and back of the arm just below the wrist. However, we may not be able to get access to the material working talent needed for this, so a ipod holding case or glove with superglue will likely be the emerging solution for this.

We use a simple Processing sketch to send the Microbit data over OSC, and so far there have been no problems with this other than issues with concatenation of Kinect & Microbit OSC data in Wekinator (discussed below).

DTW Algorithm

The DTW algorithm can't handle too many data points as it will confound the regression. Including irrelevant data points is always a bad idea in Machine Learning. We chose to only send information regarding upper body joint positions for the basketball throw, allowing the DTW algorithm to operate more successfully. This also eliminated the worry that the Kinect might detect the lower joints incorrectly. Because we angled the Kinect upwards to detect where the user's arms are at the height of the throw, we usually missed out on accurate data for the knee and lower leg positions during training. Eliminating this data as a factor greatly increased our accuracy.

We began the project by using Wekinator to perform the DTW algorithm, and categorize movements into "idle" or "throwing" for testing. However, Wekinator was a significant hassle and inconvenience for our particular project. Since we were streaming data from both the Kinect and Microbits over OSC, it made using Wekinator too complex. Wekinator cannot separate input types into two different DTW instances in a single Wekinator project, and expecting the user to run and understand two different open Wekinator projects was simply too much. This setup also did not allow us the customizability we needed to create a streamlined user interface. We needed to be able to set up our own instruction screen, different training modes, and ways to program in audio feedback to give a pleasant chime when the user succeeds at a movement. All things considered, continuing on with Wekinator was not an option.

To resolve this, we replaced our use of Wekinator with a custom desktop application (explained in more detail later), and made use of an open source Python DTW algorithm. Having access to the source code of the Dynamic Time Warping will allowed us to perform whatever tasks we wanted with it, such as analyzing the Microbits and Kinect input separately.

Custom GUI

We created a Custom GUI to make our desktop application easy to operate for the end user. This introduced far more complexity than we had at the beginning, but was worth it in the end for the customizability.

We used PyQt and Qt Designer to graphically create the user interface, using the drag and drop functions of Qt Designer to construct the appearance of the program without needing to hardcode the locations and object types of the buttons. While this made the artistic design of the pages of the application easier, it also introduced new problems. All of us are new to PyQt, so understanding how it worked was a learning process, and introduced new unknowns to the production pipeline. PyQt uses an xml file generated by the Qt Designer editor to set up objects for the Python script can interact with. We chose to write our application in Python, so this simply meant loading the xml into our main Python program and integrating it with the DTW algorithm, and any other custom functionality we wanted to add. Though the loading of the xml is simple, we had to experiment with a significant amount of trial and error to learn how to properly interact with objects created in the Designer app through a script. Through the use of online examples, we got this part online and ready to go.

Once this was resolved, we had to properly integrate the DTW algorithm with our Python script that loaded and ran the PyQt application. We ran into some unforeseen problems with the way PyQt handles button presses and other user operations. We needed to continuously stream data while a button was pressed and held down, and then stop the stream of OSC data into the DTW once the button was released (just like the press-hold-release setup in Wekinator). With exploration of button attributes in PyQt, this has been largely resolved.

Integration

The most recent challenges have arisen with putting all these disparate components together, so they work seamlessly as parts of a whole. We have a Processing script that merges Kinect and Microbit OSC messages, concatenates them, and sends them to the pipe, which is read into the DTW code. We are working on tinkering with the sysout stream so that we can read data without having to rely on OSC once the initial Kinect and Microbit OSC data is read in and stored.

Some issues have arisen from the nature of the DTW algorithm we are using, as well. DTW is unable to handle arrays of different sizes, and this has made some of the low level integration more difficult because of how we have everything set up.

Overall, the project has been generally successful so far, and is heading towards completion on schedule, retaining most of the core features.

Future Work

Remaining Challenges

Some challenges presented by this application scenario still linger, and would need to be addressed in future work. Due to time constraints, and because solving those more advanced challenges would be outside of the scope of the assignment, the following issues are still in need of solutions.

For instance, none of our team members have experience with professional personal training or coaching, and we did not have the financial resources to discuss these topics with a professional in-person. We followed a makeshift solution of attempting to gather as much free data as possible in order to minimize the risk of errors appearing in our training set while still keeping initial costs low. This worked for our limited application of only focusing on basketball throws, but some way around this lack of professional consulting would have to be addressed in the future.

In a worst-case scenario, our project may also pose a health hazard if it provides feedback which places the user in a position that threatens their overall well-being. For example, if the system consistently mis-identifies a user's pose and suggests a method to "fix" it which actually proves damaging in the long-term to their spine or back, our project could be rightly blamed for causing an even larger issue than simply their lack of proper posture. Besides the obvious physical harm to the user, any published version of our system could result in a liability to ourselves and our team. This is not an issue in the current version, which is implemented only as a pilot test system, but is a serious long term concern if this project is to be expanded.

The risk of injury, while small, is inherent in any exercise system. In order to help our users make educated decisions about when they should stop using the motion capture system or take a break from using it, a final product would include information warning potential users about indications that they should cease to use it. These indicators include signs such as physical discomfort, nausea, dizziness, etc. Disclaimer statements such as these are common in commercial exercise systems and would prove valuable to a final release version our system in order to protect our team from liability.

Another concern is the need for our system to store personally identifiable information about each user, some of which may be health related due to the nature of our program's design. Even the most seemingly inconsequential health information is governed under strict information storage laws and our data storage policy will have to reflect that. In order to make sure that this information is stored as securely as possible, future iterations of our work would have to use the most up-to-date database and storage software available. There are a variety of database security checklists that exist for this purpose. For instance, NIST is a source that provides up-to-date information security tips to the public at no cost. These resources would be necessary in order to release a "security guaranteed" publishable version of the project.

Additional Features

If the above basic security and health issues can be resolved, then there is room for future work to expand on our project with additional features. Many of the below features were planned for, but never implemented due to time or other constraints.

An obvious limitation of our system is the fact that it is currently geared towards only a single sport. We chose the basketball throw because we were familiar with it, and it was a good way to implemented our Microbit accelerometer setup. Future work would, of course, expand on the number of supported sports. We would need to make the base system itself more flexible as well, to allow sports to be added that don't just focus on the upper limbs. Our current system is heavily biased towards the upper limbs.

Once the system has more supported sports and different movements within the same sport, there would need to be a way to distinguish which movement the user was working on. Instead of forcing the system to guess which movement the user is going for, future work could use a speech recognizer to trigger the system to look for a particular gesture. For instance, let's say we install a more advanced version of the system on a basketball court. The user speaks the phrase "free throw" into a microphone, and the system knows to spend the next minute or so analyzing their movements for this gesture. Rather than confusing the movement for similar throws, the system will know exactly what to look for. The current system sets the motion ahead of time, and assumes the user is always performing a basketball throw.

An additional problem that our system does not address is the chance that the user, even with a coach present, may be exhibiting incorrect form. In the interest of customizability, our final product trains on the user exhibiting correct form, and relies on that information to help them reproduce the form later on. However, it may be difficult for a beginner to display form that is, in any sense of the word, correct. We don't want to help them form bad habits by having them reproduce their own incorrect movements.

So, another future goal is to make the system two-tiered. In order to enforce customizability, the user would still have the option to record their own movements under a coach's supervision. However, in order to prevent bad habit formations, a model would also be trained on professional movements. It is more difficult for a user to match these movements perfectly, but it would be a great way for them to measure themselves in comparison to a consistent standard.

Paying a professional once to give motion capture data to the system and then distributing it to potentially hundreds of users through a virtual coach is a modern, innovative way to revolutionize personal training. This is perhaps the most useful possibility for future work, and care should be taken to explore this option.

Additionally, the system as it is may be difficult for everyone to understand. Right now, the UI displays information about movement correctness using the raw percentage displayed by the DTW algorithm. Having the system give the user live feedback in the form of an audio signal representing how well they replicated the correct motions would greatly improve understandability. An unpleasant sounding audio signal cues the user to try again, and vice versa. This would allow the user to gauge their own progress, and improve their technique on the spot in response to audio instead of needing to wade through percentages.

However, even with audio feedback, a user may not understand what went wrong. Audio feedback with only a “positive” or “negative” meaning is too unclear, but so is a system that uses different audio cues for different joints. The former is too simplistic, and the latter too difficult and time consuming to learn. As a result, the audio feedback should ideally be combined with visual feedback to provide reinforcement regarding the meaning of each audio signal.

Such a future iteration of this project would display a very obvious visual cue to the user: a full blown “virtual coach,” who represents the model gesture. The user’s movements are motion captured and matched onto a virtual avatar. The virtual avatar moves in real time, and acts as either a mirror, or a way for the user to watch themselves from behind, or from a new angle - depending on their preference settings. Standing next to the user’s virtual avatar (or overlayed over it in three dimensional space), the virtual coach will display the correct movements for the user to follow along with and try to match. In “personal” mode, the virtual coach will display the user’s best attempt at the motion so far, as chosen by a coach from the user’s attempts. In “pro” mode, the virtual coach will replicate the movements of a professional. Using regression or other strategies, it is possible to map from the user’s body type to the professional’s body, to get a more accurate measure of how well the user is copying the professional. This method would need to take into account their unique body type and movement style.

The issue of mapping from one body type to a very different one in the machine learning algorithm is a difficult challenge to solve. This is one of the reasons we did not implement this option, along with the difficulty in constructing and optimizing 3D models.

There is much room for future expansion of this project, and many different avenues a more advanced implementation could explore. We hope this pioneering work will open the doors to future exploration within this field. Our conclusions regarding the likely future of sports teaching using motion capture technology are discussed below.

Conclusion

Overall, the project was a success! We are well on our way to a viable product, and have demonstrated that with simple technology, a primitive Motion Capture System for Sports Tutoring is possible to create. Such a system can also be made accessible to the common user, if enough care is taken on the programmer's end to integrate everything for the user.

This project demonstrates how any form of MoCap is a tradeoff between accuracy and ease of use. The Kinect represents one side of the spectrum, as a device that has limited accuracy but is very easy to use. On the other side of the spectrum is the Ipi MoCap studio, a highly accurate skeletal tracker that does not operate in real time, and is difficult to obtain and calibrate. It also requires a more complex setup, because it requires multiple cameras, and for each camera a tripod and active USB 2.0 extension cable is required for best results. Even further on that end of the spectrum are the high cost motion capture studios implemented by professional movie studios. Those operate in real time, but require a huge studio and expensive motion capture suits to implement.

The greatest concern here is that it is difficult to find a setup that will allow for sports motion capture to be used by the individual in a feasible manner. Our project has demonstrated that it is possible to create a cheap sports trainer using skeletal data, but is it practical? Our program can recognize using DTW that someone is performing a throw vs. standing still, and can sometimes recognize that the throw is "less likely" to be in the throw category when the motion is performed improperly. However, mistakes in sports techniques are often small and hard to perceive. Even with the Microbits as a supplement, our DTW algorithm cannot provide feedback to the user with enough accuracy to guarantee they are training themselves properly. This is largely because the Kinect fails to be accurate in so many use cases. A user may pick up bad habits from our system if it fails to recognize what would be obvious flaws to a human trainer.

As a result, we conclude that though the MoCap system is possible to create, it needs to be expanded in the future to make it feasible to use. A single Kinect fails to be accurate enough to provide trustworthy feedback. Future work should aim to use multiple Kinects, or find some way to program a camera system with real time streaming using cheap homemade MoCap suits and open source software. Paid software is not an option, unless this project became feasible as a business model, and we could create our own software and charge users for installation of the system.

We believe that successful home training systems will be created in the future, as camera systems become cheaper and more effective, and computer processing continues to increase in speed. Similar work to ours has been completed using MoCap for sports motion analysis (Kim, 1997; Tamir, 2008) and even physical rehabilitation (Metcalf, 2013; Chang, 2011). The Kinect in particular has already been used to help young adults with motor disabilities improve their condition (Chang, 2011).

Though a complete system with proven practical use is beyond the scope of this class, this pilot study has helped formed a picture of what could be. With enough improvements in MoCap technologies and cost, an inexpensive home system for tutoring in physical techniques could be just around the corner.

Research Bibliography

Live Document Link

Chang, Y.-J., Chen, S.-F., & Huang, J.-D. (2011). A kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities. *Research in developmental disabilities*, *32*(6), 2566–2570.

Criminisi, A., Shotton, J., Konukoglu, E., et al. (2012). Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision*, *7*(2–3), 81–227.

Grove, R. (2012, October). *Ipi soft dmc version 2.0: Affordable motion capture for everyone.* Retrieved from https://www.renderosity.com/ipi-soft-dmc-version-2-0-affordable-motion-capture-for-everyone-cms-16365

Han, J., Shao, L., Xu, D., & Shotton, J. (2013). Enhanced computer vision with microsoft kinect sensor: A review. *IEEE transactions on cybernetics*, *43*(5), 1318–1334.

Ipisoft. (2014, February). *Quick start guide for multiple playstation eye cameras configuration.* Retrieved from http://wiki.ipisoft.com/Quick_Start_Guide_for_Multiple_PlayStation_Eye_Cameras_Configuration

Ipisoft. (2018). *Ipi soft motion capture for the masses.* Retrieved from http://ipisoft.com

Khoshelham, K. (2011). Accuracy analysis of kinect depth data. In *Isprs workshop laser scanning* (Vol. 38, p. W12).

Kim, C. H. (1997, December 2). *Method of training physical skills using a digital motion analyzer and an accelerometer.* Google Patents. (US Patent 5,694,340)

Metcalf, C. D., Robinson, R., Malpass, A. J., Bogle, T. P., Dell, T. A., Harris, C., & Demain, S. H. (2013). Markerless motion capture and measurement of hand kinematics: validation and application to home-based upper limb rehabilitation. *IEEE Transactions on Biomedical Engineering*, *60*(8), 2184–2192.

Moeslund, T. B., & Granum, E. (2001). A survey of computer vision-based human motion capture. *Computer vision and image understanding*, *81*(3), 231–268.

Papadourakis, A. G. (2013, November 19). *Motion capture and analysis.* Google Patents. (US Patent 8,589,114)

Smisek, J., Jancosek, M., & Pajdla, T. (2013). 3d with kinect. In *Consumer depth cameras for computer vision* (pp. 3–25). Springer.

Tamir, M., & Oz, G. (2008, August 14). *Real-time objects tracking and motion capture in sports events.* Google Patents. (US Patent App. 11/909,080)

Zhang, Z. (2012). Microsoft kinect sensor and its effect. *IEEE multimedia*, *19*(2), 4–10.