**Shimmer IMU Impact and Rotation Research Project**

**Introduction**

Investigate gathering real time data from Shimmer3 IMU device for use with 3D model rotation and data analysis from player movement in a training scenario. This project targets Rugby specifically, and is intended for use by a coach for assisting players to perform, learn and practice safe tackling in a training environment.

**System Requirements**

* PC, Laptop, Mac.
* Windows 10, Mac OS 10.13 or later.
* Minimum 1 human to interact with the application.
* Shimmer3 IMU and strap.

**User Perspective**

A user manual has been produced to guide the User through the UI of the application. The application is intended for use by coaches who wish to record live data from the movement of their players during training sessions.

Specifically, this application targets *Rugby* tackling and a means to prevent injury by correcting the angle of attack of a player during a tackle. The User will be able to record a specific set of data while the player performs a tackle, and review this post tackle with the player, to gain valuable perspective and correct any bad habits or a potentially damaging approach.

The application will record the angle and rotation of the players spine during the tackle for post analysis with the coach. This can be replayed again immediately, saved for a later review, and saved as a log of a player’s progress through training, which can be reviewed conveniently by loading the session into the replay feature of the application.

What the User sees is a 3D model representing the player’s spine. In this application we represent the spine with a straight line, this model can be updated during any further development. Once the Shimmer device is streaming data to the application, the 3D model will rotate in real time via Bluetooth connection (range 30ft). Recording the data as outlined in the User manual, will record until stopped and can be saved and replayed conveniently. The replay feature will play back the session which has just been recorded, with an option to load a previously saved session from a players training file etc.

**Developer Perspective**

The application is developed with Unity3D community edition and C#. During the development we used the SDK which came with the Shimmer devices which handles streaming data from the IMU devices to the Unity application.

<http://www.shimmersensing.com/>

ADD DETAILS – a concise version of the architecture ?

**Development Cycle**

As part of the research it was important to investigate different approaches to the project. There were multiple approaches to the project during the early stages of development. These gave us a clear idea of the requirements and to some extent, the limitations (see Limitations and Bugs section) of this project.

**Shimmer C# SDK**

The initial stage of development involved working with the SDK and configuring the devices with the included software applications. It is possible to export and import device configuration settings, to maintain consistent data between devices. This was a few weeks of trial and error which lead us to investigate some other options as the included software did not have the features we would need to build a prototype application for the training sessions. We did however make our own stripped down version of the Shimmer library, for use with the other projects.

**ShimmerRT Library**

This project is a C# .NET 4.6.1 class library (.dll) which we have developed and it consists of code taken from the official Shimmer C# SDK. By creating this library, it enabled us to keep the code used to access the Shimmer device separate from each of the research projects which we would undertake. If this library needed to be updated, it was then a simple task to update each project with the new DLL and push this to git rather than updating this code on a per-project basis.

To keep the testing and debugging of this class library simple and manageable, a small Windows CLI application has also been included in the solution. This CLI application allows a user to connect to a Shimmer device by typing in the COM port which the Shimmer device is connected to via Bluetooth. While it is not ideal to require the user to know the COM port, this was very helpful in the initial stages of retrieving data from the Shimmer. This COM port issue is something that we would overcome at a later stage using a workaround detailed below in the section “Shimmer Discovery Windows Application”.

Github Repository – Shimmer Library: <https://github.com/SerjiVutinss/ShimmerRT_Library>

**Unity Research**

The Shimmer SDK provided us with the means to stream and read the data from the IMU device. Next step was to bring it into Unity and move an object. This stage fleshed out the use of the gyroscope, accelerometer and quaternion data from the device to manipulate an object in a 3D environment.

Github Repository – Unity Prototype: <https://github.com/SerjiVutinss/ShimmerRT_Unity>

**Thalmic Labs**

Further development in rotating objects with gyroscopes and quaternions lead us to research the Thalmic MYO’s gesture control armband and SDK. We had access to an armband through college and the development kit is a free download:

<https://developerblog.myo.com/author/thalmic-labs/>

This was a valuable undertaking as we could analyze and understand another approach to rotating a 3D Model using Quaternions.

We used Unity3D as our platform, the MYO example included in the SDK was also in Unity3D. We interrogated the MYO SDK and developed a unity scene to test our rotation and data gathering with the Shimmer3 IMU.

This version contains the most testing and is the platform which was used to develop the current version of the project. We experimented with a simple 3D model and a simple humanoid rig imported from Blender. This returned good results in terms of accurate rotation and highlighted some issues with the model drifting after the device comes to a stop.

Github Repository: <https://github.com/d-gallagher/Myo_Shimmer_Research>

**Blender**

We built a simple humaniod rig in Blender for this application and tested the rig as a possible model for the application. With the armatures from the rig it is possible to map the shimmer to a specific bone rotation like the spine, arm or a leg for example. The box model is the primary feedback for the UI. It works better as a simple display. The rig is a bit more complicated to implement how we would like it so it will remain dormant in the project for future development.

**WPF Application**

We looked at a stand-alone C# application using WPF to build out the GUI. This was successful as we had rotations working and some UI feedback for the user to show axes current rotation. We approached it from a design perspective at first, building out the 3d model in xaml and rotating with some C# code snippets. We later decided to build the model using C# as it was easier to colour the sides on the model that way, and hence to see the model rotating.

This iteration of the project was inspired by the following project which was developed for the MYO armband: <https://elbruno.com/2016/07/28/myo-working-with-the-orientation-and-gyroscope-in-c/>

While we had accurate rotation from our model in the WPF application, a decision was made to pursue the Unity 3D further as it supports importing models from Blender to use in scenes. It became clear during a meeting that during future development, a human 3D model might be well suited to the playback feature and perhaps the real time streaming view of the shimmer device. Unity 3D offers a lot of UI features and a physics engine also which were added bonuses for menu development.

Github Repository: <https://github.com/d-gallagher/ShimmerRT_WPF>

**Hololens Unity/UWP Application**

We considered deploying to the Hololens as a part of our project. From our research we realized a substantial portion of the source code would have to be refactored and re-written. Building the Shimmer SDK as a UWP class library for example.

System.IO.Ports is unlikely to work on UWP due to app sandboxing restrictions, this is important as our shimmer device depends on this class to function.

An alternative SerialCommunication Class:

<https://docs.microsoft.com/en-us/uwp/api/windows.devices.serialcommunication>

has been recommended on various forums as a work-around. Additionally, all Threaded code would have to be swapped out for Tasks as System.Threading is unavailable in the UWP platform. We investigated these workaround options and decided that given the short time span of our project and the work required to adapt everything for the Hololens, our time would be better spent focusing on the windows-based application.

We believe the Hololens, or an Augmented Reality compatible version of the unity application is viable for future development and would be a suitable implementation for the mobile requirements of both the coach and players partaking in the training sessions.

**Uduino Research**

We looked at an Arduino solution to a similar project to ours. There wasn’t enough time to set up a build and test the actual Arduino device but this was an interesting project which implemented 6 degrees of freedom rotation from a small device to a 3D model. In the example project we looked at, the device was using a wired connection which would not suit the training sessions we were working towards.

**Shimmer Discovery Windows Application**

As mentioned above in the Shimmer RT Library section, we faced an issue regarding how the user of any of the projects detailed above would connect to the Shimmer. Specifically, a user would need to know the COM port to which the Shimmer is connected via Bluetooth. This COM port is not assigned consistently and will be different across different machines for the same Shimmer device. It also appears that it is liable to change occasionally upon connection of the device to a previously used machine.

This issue raised a big problem regarding ease of use as it would require the user to look up the COM port through Device Manager in Windows. To combat this, we developed a small Windows Console application which retrieves any Shimmer devices registered in Windows and their assigned COM ports. The application saves this data to a JSON file which is then read by the Unity application and is used to populate a dropdown list. The user may select the relevant Shimmer from the dropdown and connect to it with little effort.

This application is run by the Unity project on start up, so it is still a requirement that the user has both enabled Bluetooth and paired the Shimmer device before running the application.

**Impact and Rotation in Unity**

The final version of the research is a consolidation of each of the above. We built an intuitive UI to guide the User using clearly named buttons and red/green colour feedback when the device is active.

Connecting, recording and playback are available from a drop down menu, each a convenient panel with buttons for each activity. A comprehensive User Manual is available to guide a User through the UI.

The application handles files to load and save training sessions. Sessions can be played back from a file or from a recording. They are saved as csv files for easy use in other applications like Excel.

The source code is well commented for clarity and readability.

*Add details about the device connecting and any other details I’ve missed o\_0*

**Architecture of the Solution**

*<<<<<Fill in details here if ye think it’s worth it, doesn’t need to be to mad I think. No point in repeating what’s mentioned in the Development Perspective either so could be a good idea to maybe just do the dev perspective? Whatever ye think anyway, if both is no hassle then both 😉 >>>>>*

*Unity IMU Impact and Rotation - Arch (concise)*

*Basic run through of Unity\_IMU\_ImpactAndRotation...*

*Shimmer RT Library - adapted code from the supplied Shimmer C# SDK. We have only exposed two classes and one interface from this library:*

*\* Shimmer3DModel class - a custom POCO class which is the representation of a record of data output by a Shimmer device. While the original SDK manipulated the Shimmer data by using a simple array of 64-bit double-precision numbers, we made the decision to use a POCO so that it was clearer to us which data point we were accessing at any particular time, i.e. shimmerModel.GYRO\_Z instead of shimmerData[24], etc. This class also contains a static method to create an instance from an array of double floating point numbers, the data structure used in the original SDK.*

*\* ShimmerController class - this class is heavily based on a class in the original SDK with some very small changes, in particular, its constructor takes an IFeedable interface as a parameter. In this way, it is possible to pass data back to the class which has instantiated the controller. In the case of this particular Unity project, the caller is the ShimmerDevice unity script. The fundamental purpose of this class is to interact directly with the Shimmer device/unit - it contains methods to Connect/Disconnect, Start/Stop streaming, etc. Due to the fact that the original class is 1200+ lines of code, much of it has been left intact in the abstract class ShimmerControllerBase, which this class (ShimmerController) extends. All of the higher level code is contained in this sub class.*

*\* IFeedable interface - an interface which contains one method signature - UpdateFeed(List<double> data). We use this interface as a means of passing data between the ShimmerController and the class which instantiated the controller. The decision to use an interface was made early in development since we were aware that we would be working on mutiple different projects/frameworks in C#, so a common interface made architectural sense rather than a concrete class.*

*ShimmerDevice.cs - this is the Unity representation of a Shimmer device. It contains a reference to a ShimmerController instance which is used to interact with the Shimmer unit. This class also implements the IFeedable interface and thus can receive data from the Shimmer. This data (in the form of Shimmer3DModel.cs objects) is placed into a Queue data structure as it is received.*

*ShimmerJointOrientation.cs - this script is attached to the 3D Unity model. It contains a reference to the ShimmerDevice.cs script (assigned in its Start() method) and has access to that script's Queue data structure. As data is placed on the Queue, it is dequeued in this script's Update method and the data is used to manipulate the transform of the 3D model. This works well since Unity runs at ~60 frames per second while far fewer data records per second are received from the Shimmer unit.*

**Testing**

Testing has been conducted constantly to determine if our device was reading data, rotating in the correct axes, replaying data from memory and saved files, saving data to a file, testing buttons in the UI to ensure they do what they’re designed to. We have worn the device to test the distance of operation, it’s accurate while in Bluetooth range and will disconnect at about 30 feet. We have not had a chance to bring the device to a training centre for more testing in a tackling scenario.

**Limitations and Known Bugs**

**Bluetooth range**

* Range is ~30ft which is a minor limitation with regard to tracking players during a real match. However for the context of this project, where the coach will be within 30ft of the player this is not expected to cause any serious issues.
* This is expected behaviour and was a known limitation from the beginning of the project.

**Lag from the device to the Shimmer Capture application.**

* We noticed a lot of lag (about 7 seconds), when using the shimmer rotation example provided with the capture software. This would be a critical failure however in our Unity application there is no latency or lag so the data stream is accurate and is not causing any issues.

**Drift in 3D Model after Device Stops**

* As mentioned in the Thalmic Labs section, we had unexpected behaviour from the 3D model while streaming the data.
* This issue is inconvenient but since the device is accurately streaming data while in motion it does not affect the results gathered.
* During research it was unclear if this is definitely a hardware issue or a software issue, although having tested different software approaches we are inclined to think it’s the former.

**Hololens, Serial Ports and Project Scope.**

* During the Hololens project we discovered some difficulties in using serial ports. This would entail refactoring the project and ultimately it was decided to table this for future development as it would be a long and involved task. The following points document this discovery.
* <https://forum.unity.com/threads/hololens-bluetooth-ports.497320/>
* <https://answers.unity.com/questions/1467623/type-or-namespace-not-found-in-namespace-systemio.html>
* <https://stackoverflow.com/questions/36380925/how-to-write-serial-data-to-com-ports-with-universal-windows-application>

**Recommendations for future Development**

Developing the application for Microsoft Hololens or other Augmented Reality / Mixed Reality headsets is possible with more time and planning, and would be a viable implementation of this project. As a mobile solution this would suit the training environment and with future developments in smart glasses and powerful smart phones it’s reasonable to consider this project as a mobile app rendering on smart glasses.

With our research into using a humanoid rig and mapping a player’s movements onto the rig, we believe there is promise here. Our deadline didn’t allow us to test multiple shimmer devices but this would lend its self nicely to the blender rig concept. We discussed the possibility of having a player wear multiple devices, mapped to the joints of the blender rig and returning a visual representation of a players body. This leads to further questions however with a player wearing multiple devices which could be damaged Where a player is tackling in a training scenario it might work but it is unlikely to scale well for a real match playing scenario.

Future development can introduce extra feedback to the user for analysis which we did not have time to develop.

* UI feedback to show acceptable, risky and dangerous rotation of spine during tackle. This will require configuration along with the correct metrics from the sports research dept. A simple green, orange, red colour change in the model to portray the rotation as safe, risk and dangerous would be a good start.

**Conclusions**

It is clear that further research, planning and development will be necessary to fully realize the potential of the shimmer devices. We have been able to demonstrate that the project is promising and can be used to track the angle of incline and the rotation of a player’s spine. The devices are reliable to work with, good battery life and can be configured from a single set of configuration files which means multiple devices will all be configured the same, data will be consistent.