Sensei: Sports Assistant

Daniel Jang, Debosmit Ray, Jin Choe, Matthew Reynolds

CSE 477, Spring 2015

ABSTRACT

Many sports players spend more of their time practicing than actual playing a game. With this motivation, we wanted a device to help assist the players during their practice sessions. We present Sensei, a device that has a built in scorekeeper, and a motion analyzer. With three different users we analyzed motions. After analysis we used our algorithm to determine how similar the motion is. This similarity we call our consistency score, the closer to 100, the more consistent the motion are. To evaluate our method, we tested the common case and a few edge cases. The common case being a natural swing, stroke. The edge cases we tested were static motion, very slow, and very fast motions. Our result was true to what we expected. Similar motions received high consistency score whereas dissimilar motions resulted in low consistency score. In addition we implemented an interface for the user to view their motion in a graphical form. The graph plots motion, where x-axis is time, and y-axis is the magnitude of acceleration. Below this graph we display the maximum acceleration for each motion. We have also implemented a scorekeeper which supports tennis, basketball, and golf. Furthermore we implemented a system to improve your sport's experience. Sensei will help make a user more consistent in sports and be a scorekeeper for them.

Author Keywords

Sports; inertial measurement unit; wearable; dynamic time warping,

ACM Classification Keywords

C.0; C.3; I.1.2; special-purpose and application-based systems, microprocessor and microcomputer applications, signal processing systems, real-time and embedded systems, computing methologies, symobolic and algebraic manipulation

General Terms

Algorithms; Design; Human Factors; Performance

INTRODUCTION

People play sports everyday and is a healthy part of your lifestyle. We wanted to promote people to play more sports. However many people don't enjoy playing a sport because they aren't good at it. Our goal is to help anyone become better. One method of doing so is to get a personal coach. However, getting a personal coach for a training session is old-school and very expensive. In order to solve this problem, we came up with Sensei. Sensei is a wearable device that can act as your personal coach. Not only will it help improve your game, it can also keep scores for you. How often does it happen that you forget the score of the game? For many players, this is very common. To put an end to a problem, we implemented a feature that acts as your scorekeeper. With Sensei, you will never forget the score of a game.

Consistency feature

Our wearable devices will help you in sports. Sensei is a device that will quantify your motion during a sports session. Whether it be basketball, tennis, golf, or badminton, Sensei can help make your sports life easier. Sensei is designed to measure your consistency, record the good shots and bad shots. It will deliver then a statistics to show the user how they are performing for a particular stroke.

Acceleration of motion

Sensei also has the ability to show the user the graph of their motions. This graph is represented where the x-axis is time and y-axis is magnitude of acceleration. We find that many people want to quantify how powerful their stroke is. With Sensei that is possible.

Scorekeeper

Lastly, Sensei implements a scorekeeper. Although it is important for sports players to keep track of the score, people often forget the score of their game. Sensei can do that for you so that you can focus on the game!

BACKGROUND

The core of Sensei is essentially made up of three components: the IMU (inertial measurement unit), RFduino, and Android application.

Inertial measurement unit

The IMU (inertial measurement unit) is commonly used to measure the kinematics of a device. A 6-axis IMU uses a 3-axis accelerometer and a 3-axis gyroscope. The 3-axis accelerometer is a sensor that outputs the acceleration in the x, y, and z direction. Similarly the 3-axis gyroscope is a sensor that outputs the angular speed in the x, y, and z direction.

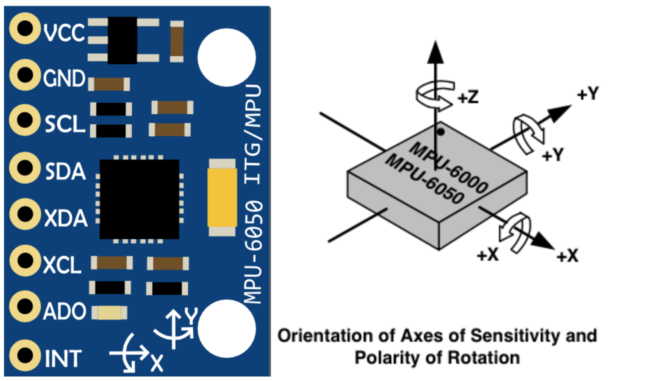


Figure 1: MPU-6050, 6-axis inertial measurement unit (*left*) and diagram of MPU-6050 with axis labeled (right)

RFduino

An RFduino is a finger-tip sized, Arduino compatible, wireless enabled microcontroller. It sports a ARM Cortex-M0 processor and has a built in Bluetooth 4.0 Low Energy module. This device is reprogrammable using the Arduino IDE. The code is written in the C language. The microcontroller's typical power supply is 3V. The device has 128kb of flash memory and 8kb of ram.

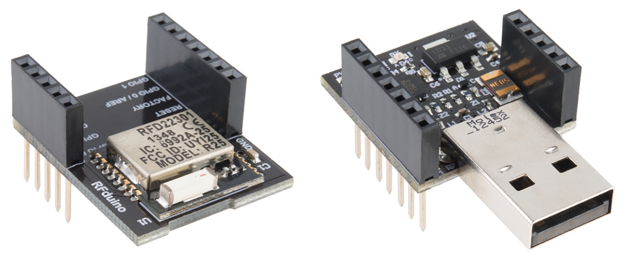


Figure 2: RFduino (*left*) and usb programmable shield (*right*)

Android

Android is a mobile operating system developed by Google. Android is used to provide interaction Sensei. In addition with the mobile phone, it is used for its processing power. The android phone processes the raw values from the IMU and performs various algorithms to compute the results.

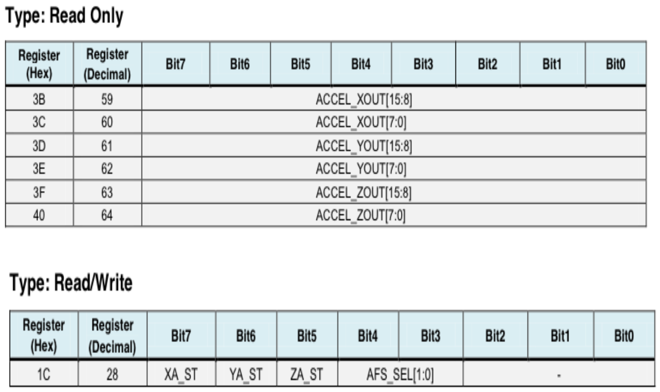
IMPLEMENTATION METHOD

Sensei has three main components: the RFduino, IMU, and Android application. The RFduino communicates with the IMU via I2C. The RFduino communicates with the Android phone via Bluetooth. Therefore the system is connected from the IMU to the Android phone.

Hardware

Configuring the IMU

To measure the acceleration and rotation of the device we use the IMU and an RFduino. The IMU uses 16-bit ADC per axis of measurement. In its default configuration, the accelerometer has a scale of +/- 2g. For our intent, we need this value to be much higher. The trade off is precision. If we increase the accelerometer range, then we are essentially decreasing the precision of our accelerometer. The largest range that the IMU supports is +/- 16g. In order for this we need to set the AFS_SEL for each axis to be the value 3, as shown in the figure below.



Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

Figure 3: Register mapping of the accelerometer (*top*) and accelerometer configuration register (*below*)

According to the datasheet, values from -16g to 16g are stored as a 16 bit 2's complement value. Its full scale is +/- 16g with a LSB sensitivity of 2048/g. This means that 2048 digital output is a unit of 1 g. For a 16 bit 2's complement, the range is -2^15 to 2^15-1 or -32,768 to 32767. A value from the sensor can be directly translated to units of g using the unit conversion of 2048/g. So for example, if the sensor reads accelX = 6032, accelY = 8382, and accelZ = 2020, we can divide values by the LSB sensitivity constant 2048 to get the acceleration in each axis in units of g. accelX = 6032/2048 = 2.95g, accelY = 8382/2048 = 4.09g, and accelZ = 2020/2048 = 0.99 g. Since we are using 2 bytes per axis of acceleration, we will need 6 bytes of data from the IMU.

Sending data from RFduino

The IMU is paired with an RFduino. The main job of the RFduino is to read the raw values from the sensor, and encode the 6 bytes of acceleration data to a string and send it to the Android via Bluetooth. We obtained the data from the IMU via I2C. After successfully reading the data we verified to make sure it was correct. We graphed the values coming from the sensor and realized a potential problem. Speed was an issue. The frequency at which we are sending data severely impacts our ability to determine whether a frame of data is consistent to another frame. At first without any delays we were getting a frequency of about 25 Hz. In other words we were sending raw data 25 times per second. This wasn't sufficient because if we are moving the device very fast, then the capturing of motion is not as smooth. We later optimized our RFduino code's efficiency. With this change, we were able to send raw data at a rate of 80 Hz. This helped smooth our data graphs and improved our precision.

Prototyping the housing unit

To enclose our device, we printed our housing unit. The housing unit has 4 button exposed on the top. The housing unit uses a "double T" design to access the button. The motive behind the double T design is used to lock the button in place so that it does not fall out. The circular pad, which is connected to the double T support material, can be pressed exteriorly. The flat-end part of the double T support is pressed against the device's button. This allows users to press buttons from the exterior.

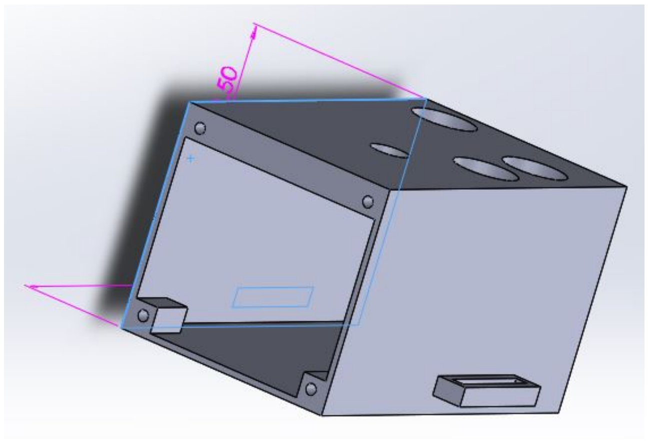


Figure 4: Prototype case for the device

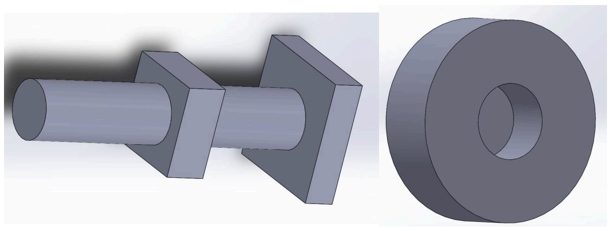


Figure 5: Double T design for button (*left*) and exterior button for the housing unit (*right*)

Software

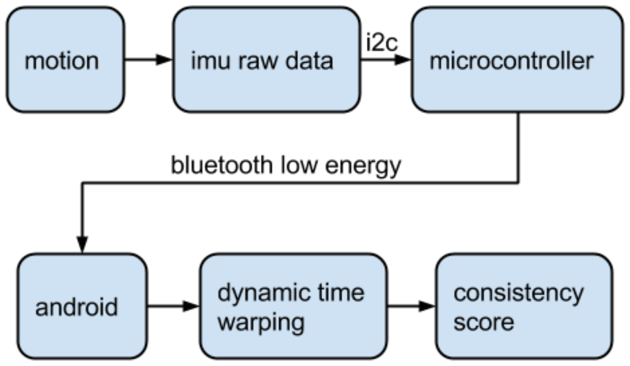


Figure 6: Data flowchart, from the user's motion to the consistency score

The Android phone handles most of the processing. When the motion button is pressed on the RFduino, the Android process the data until the button is pressed again. It decodes the data from the RFduino and stores it into a Motion Object. When Motion data is made up of an array of data coming from the accelerometer. Once the user has done a variety of motions, when the user swipes to the consistency tab, it calculates the score by comparing the motion objects that were created. The biggest problem with this is the method of comparing. How should you compare sets of data? Without some measure of warping the data, it is very hard because the data has a component of time that we do not care too much about during comparisons. A stroke or motion that is similar but starts later in time should be treated without factoring the time. The solution to this problem was dynamic time warping. Using dynamic time warping our output is the path cost from one motion to another motion. The lower the path cost, the more similar the motions are. This is the core of our consistency score rating. We also considered the noise. Even when the IMU is static, there are some fluctuations in the data. This noise adds some error in our motion consistency algorithm. We balance this noise with a initial threshold for our path cost. This initial threshold is minimal and is enough to cover the static noise we are getting from the data.

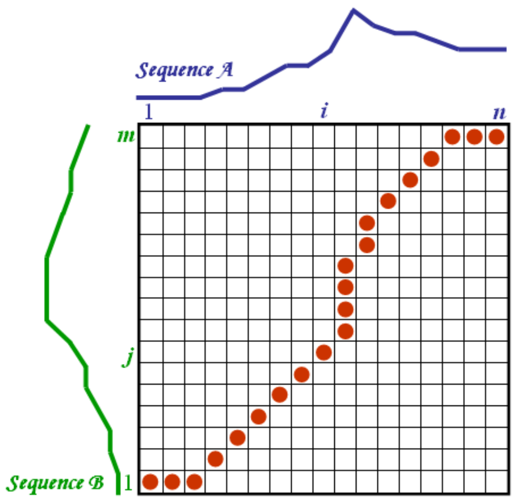


Figure 7: Dynamic time warping algorithm grid

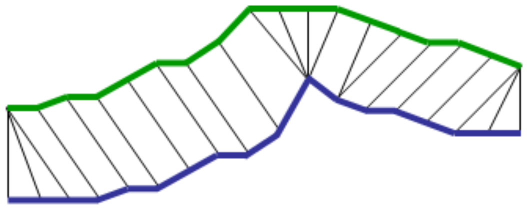


Figure 8: Visual representation of DTW alignment

RESULTS

To evaluate and verify our system, we tested the common case and the edge cases. One of our edge cases was the resting position. In this position two or more motions should achieve a very high consistency score. Other edges cases include very slow motions and very fast motions. All group members participated in performing different motions of varying speed and time. The data we collected was processed in our algorithm and we debugged and analyzed every stage of the algorithm to make sure each was functional. Our motion was processed into a motion array which was then dynamically time warped to calculate the costs paths. For each case we tested, we compared this cost path to our modified threshold values. We determined what was reasonable ratio of cost path to threshold and determined a percentage based on that. We adjusted the algorithm based on our tests.

The cost path increases as the magnitudes of the signal increases. We need a way to normalize this data in order to compare the data. Instead of normalizing our data, we adjusted the consistency threshold value based on the magnitudes of the signal.

To test DTW we first generated vectors to compare. We made a function to make these vectors similar by a percentage. This percentage can be changed by the tester. We tested intensively by running these vectors through the algorithm. Our output cost paths should reflect based on the percantages we chose. For example a percentage of 90% similarity should be giving a lower cost path than one with 80% similarity. We got a set of varying costs paths and verified that the smaller costs paths are associated to higher similarity, and vice versa.

CHALLENGES

Housing challenges

One of our biggest challenges was lining up the button holes from the housing unit to the buttons on the device. Not only was it hard getting precise measurements, but the 3D printing error was a big factor. Since we are in units of millimeters, the error on the 3D printing could not be ignored. Many adjustments to our 3D model has been made.

Another

Consistency algorithm challenges

IMPROVEMENTS

For a better user experience, replacing the push buttons with a touch sensor would be necessary. Another improvement to be made is to get rid of a start and stop signal. We hope in the future our device just requires one tap to start a sport session. Throughout the session the user can make multiple motions and the device will detect what was considered a motion and compare them. This will reduce a lot of the users manual work to use the device. This can be possible with the use of the gyroscope and accelerator data. When the gyroscope and accelerator data exceeds a threshold that can be the signal that a motion is starting. When it those values get below a threshold that is when a motion is stopped. We hope we could implement this feature in the future. Also adding a small display screen can make scorekeeper completely independent of our Android phone.

CONCLUSION

Our team wanted a device that can replace the need for a sports coach. Sensei, as a wearable device, can be a portable replacement for a coach. It helps quantify and evaluate your sports session. Sensei's main motivation is to deliver the user a metric in which they can evaluate their stroke. The device will give a visual representation of their motion. We believe that a sports player should strive for consistency. Sensei will give the user a score on how consistent their motion is. The IMU measures the motion at a rate of 80 Hz and the Android phone processes the data and performs algorithms to determine the score. The consistency score is determined by applying a dynamic time warping algorithm to the sensor data.

REFERENCES

http://en.wikipedia.org/wiki/Dynamic_time_warpingDing,

http://www.psb.ugent.be/cbd/papers/gentxwarper/DTWalgorithm.htm

http://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/

http://www.rfduino.com/