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 the game. With this motivation, we wanted a device to help assist players during their practice sessions. We present Sensei, a device that has a built in scorekeeper and a motion analyzer. Three participants wore the device, and we analyzed their motions. After analysis we used our algorithm to determine how similar the motion is. This similarity, we call our consistency score. The consistency score is a value between 0-100. The higher the value, the more consistent the motions are. To evaluate our method, we exercised many different cases. Our team tested the common case and the edge cases. For our purpose, the common case is a natural swing. For our system, we identified three edge cases. The edge cases were: static motions, very slow motions, and fast motions. We exercised these cases and our result was true to what we expected. Similar motions received high consistency score whereas dissimilar motions resulted in low consistency score. In addition to our consistency score, 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. As our final feature, our team implemented a scorekeeper that supports tennis, basketball, and golf. Our goal was to provide people with a wearable device that is capable of improving their sport's experience. Sensei will improve a player's consistency and also keep score 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

Playing sports is healthy and should be part of people's lifestyle. We believe that people should be active and enjoy playing sports. However some people might not enjoy playing a sport because they are not good at it. Our goal is to provide an experience that will help people become better at a sport. The old-fashioned method of doing so is to hire a personal coach. Hiring a personal coach is not only is a hassle for people, but is also 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 be your own personal scorekeeper.

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 is 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 is a wearable scorekeeper. 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.

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.

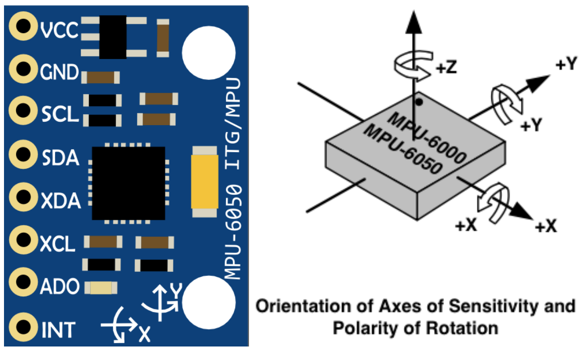


Figure 1: MPU-6050, 6-axis inertial measurement unit (*left*) and diagram of MPU-6050 with axis labeled (right)

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. With the usb shield, this device is reprogrammable using its Arduino IDE. Code for the RFduino is written in the C language. The microcontroller's typical power supply is 3V. The RFduino device has 128kb of flash memory and 8kb of ram onboard.

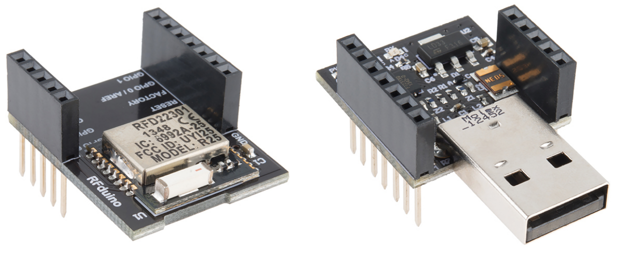


Figure 2: RFduino (*left*) and usb programmable shield (*right*)

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.

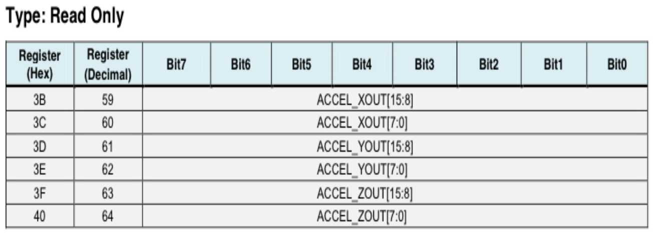
IMPLEMENTATION METHOD

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

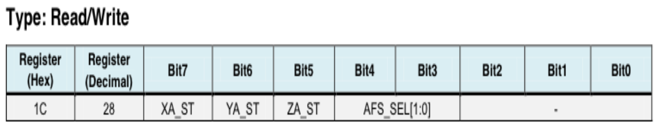
Configuring the IMU

To measure the acceleration and rotation of the device we use the IMU. 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 the range to be higher. Since the bit representation of the value is constant, increasing the accelerometer range will decrease the precision of our accelerometer. The largest range that the IMU supports is ±16g. In order to adjust the sensitivity to ±16g, we set the AFS_SEL bits to be the value 3. We locate which register AFS_SEL is using the register map datasheet as shown in figure 3. Since AFS_SEL[1:0] are located in register 1C and are bits 3 and 4, we set the value of register 1C to be, 00011000 or 0x18. According to the datasheet, values from accelerometer 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 a digital output of 2048 is equivalent to a unit of 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 converted to units of g using the unit conversion of 2048/g. So for example, if the sensor reads: accel_x = 6032, accel_y = 8382, and accel_z = 2020, we can divide values by the LSB sensitivity constant 2048 to get the acceleration in each axis in units of g: accel_x = 6032/2048 ≈ 2.95g, accel_y = 8382/2048 ≈ 4.09g, and accel_z = 2020/2048 ≈ 0.99 g. There are 2 bytes of accelerometer values per axis. Therefore, accelerometer data in 3 axis will be 6 bytes.



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*)

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 IMU data and send it to the Android via Bluetooth. We obtained the raw data from the IMU to our RFduino via I2C. The next step is to encode the data into bytes and sent these bytes to our Android application via Bluetooth low energy.

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.

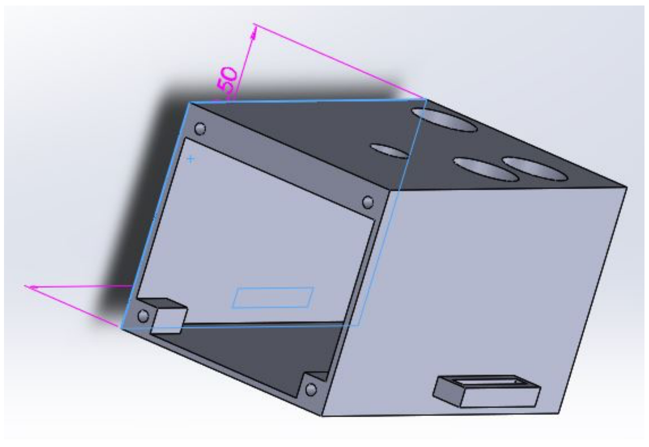


Figure 4: Prototype case for the device

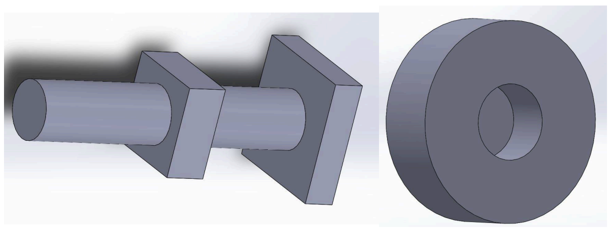


Figure 5: Double T design for button (*left*) and exterior button for the housing unit (*right*)

Software

Data flow

The Android phone handles most of the processing. When the motion button is pressed on the RFduino, the Android phone processes the data until the button is pressed again. It decodes the data from the RFduino and stores it into a Motion object. Our algorithm is then applied to all the motion objects accumulated by the user and outputs the consistency score, a rating value between 0 and 100.

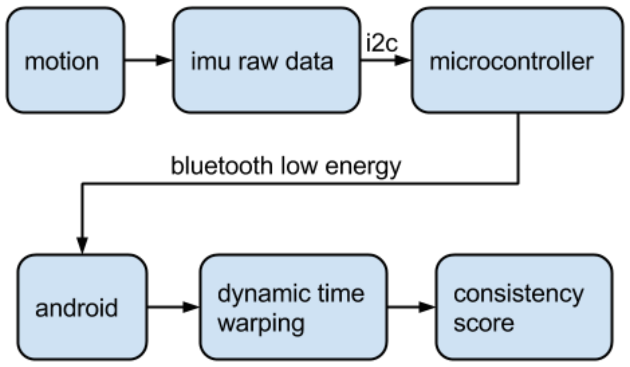


Figure 6: Data flowchart, from the user's motion to the consistency score

Detecting similarities in motion data

Once the user has done a variety of motions, the user can view his/her consistency score by swiping to the consistency tab. It calculates the score by comparing the motion objects. This is the most challenging part. How should you compare sets of data motion? What approach is optimal? Without some measure of warping the data, it is very hard because the data has a component of time that we do not want to factor when making the comparisons. Two motions that are similar but starts at different times should output as a similar motion. The solution to our 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.

Dynamic time warping

DTW is a time series alignment algorithm. It aligns two sequences of vectors by warping the time axis iteratively until an optimal match between the sequences is found [4]. To understand how this algorithm works, you can visualize a 2D grid. Each cell in the grid represents the minimum path costs to reach that cell. The cost is related to the motion vectors that are being compared. Similar values result in lower costs. This algorithm works by keeping track of the costs of the best paths to each point in the grid. Once the end point is reached, it calculates the minimum cost value by accumulating all the cost values at each point in the path. For our purpose, the value of the path cost is related to the consistency of motions. High path costs in two motions infers that the motions are not similar, whereas low path costs infers that the motions are similar. At some point we must draw a line to determine what is considered a "good" score and "bad" score. The quantitative representation of this line is a threshold value.

Considering different motion speeds

The cost path increases as the magnitudes of the signal increases. Our team considered two options of solving the issue; we must either normalize the data or consider an algorithm to adjust the threshold value. With some testing, we found the later to be more effective.

Dealing with noise

When dealing with data, noise must be considered. Even when the IMU is static, there are fluctuations in the data. This noise adds some error in our motion consistency algorithm. We balance this noise with an initial threshold for our path cost. This initial threshold is minimal and is enough to cover the static noise we are getting from the IMU.

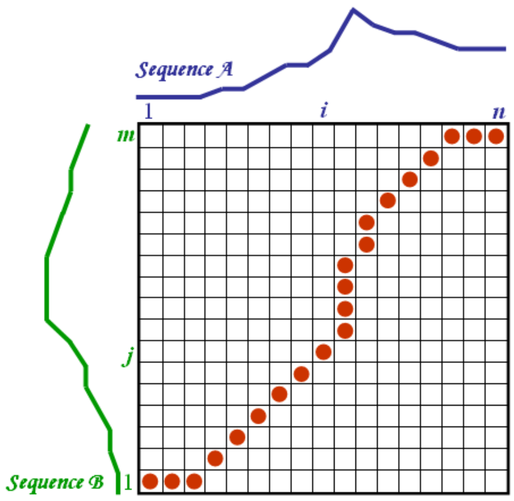


Figure 7: Dynamic time warping algorithm grid

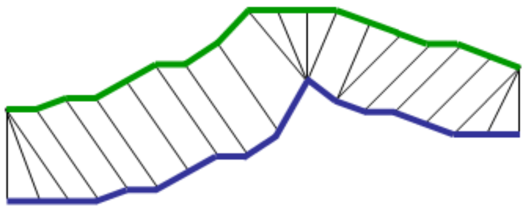


Figure 8: Visual representation of DTW alignment

RESULTS

Verifying Sensei

Testing the DTW algorithm

To test DTW we verified that closer related vectors resulted in lower DTW path costs. We first generated vectors to compare. We made a function to generate vectors that would be similar by a given percentage. We made a variety of vectors that were similar by different percentages. DTW was then applied to these vectors and we viewed the results. Our output cost paths should reflect based on the percentages of similarity of the vectors. For example a percentage of 90% similarity should be giving a lower cost path than one with 80% similarity. We verified that the smaller costs paths were associated to the vectors with higher similarity, and vice versa.

Testing the consistency algorithm

To evaluate and verify our consistency algorithm, 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. Three participants performed 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 a relation between path cost and consistency percentage based on that. Since there are many different motions to consider, throughout testing, many adjustments of the algorithm had to be made.

CHALLENGES

Frequency of sending data

After successfully reading the IMU data we verified to make sure it matches with our encoding. Next, 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 was not 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.

Housing challenges

Since none of us were mechanical experts, one of our biggest challenges was to have buttons that were accessible from the 3D housing. We looked at several designs and decided that a "double T" design would be effective for our housing unit. Although effective, installing the double T pillar design to support our device, outer housing, and button cap was difficult. Many manual adjustments were made to ensure that everything was lined up correctly. Lining up the button-holes from the housing unit to the buttons on the device was also challenging. Not only was it hard to get precise measurements, but also the 3D printer's error was a big problem. Since we most measurements were taken in units of millimeters, the error on the 3D printing could not be ignored. Many manual adjustments to our 3D model had to be made in the housing process.

Consistency algorithm challenges

One of our biggest challenges in determining the consistency algorithm was to figure out the proper way of comparing signals that vary in time. Although they vary in time, our algorithm must be able to adjust to time. If you compare the data at the same time segments, two motions that are identical, but shifted in time will conclude that the motions are not similar. Applying DTW to our motion data solved this problem.

IMPROVEMENTS

User experience

For a better user experience, replacing the push buttons with a touch sensor would be necessary. Another improvement is to get rid of the 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 setup work of the device. We believe 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 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.

Retrieving accurate and smooth data

To perfect our design, our system would require continuous data. Since that is impossible, the best we could do is to maximize the sensor's data throughput and smooth our data. Our current throughput is 80 Hz. One method of improving our results is to smooth the data. Although a rate of 80 Hz is sufficient for our product, we can improve this by smoothing out the signal. A perfect sensor would have no spikes in the data. The difference in two adjacent acceleration readings should be similar in magnitude. Since motion is continuous we would want our data to be more continuous. If we can make our IMU data have less spikes, it would improve our results. We can improve our design by applying a smoothing function to reduce spikes in our data.

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

1. Wikipedia. (2015) Dynamic Time Warping. Retrieved June 1, 2015. http://en.wikipedia.org/wiki/Dynamic_time_warping
2. InvenSense. (2015) MPU-6050. Retrieved June 1, 2015. http://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/
3. RFduino. (2014) RFduino. Retrieved June 1, 2015. http://www.rfduino.com/product/rfd22102-rfduino-dip/index.html
4. GenTχWarper. DTW algorithm. Retrieved June 1, 2015. http://www.psb.ugent.be/cbd/papers/gentxwarper/DTWalgorithm.htm
5. Hiroaki Sakoe and Siebi Chiba, "Dynamic programming algorithm optimization for spoken word recognition," Acoustics, Speech and Signal Processing, IEEE Tnransactions on, vol. 26, no. 1, pp. 43-49, Feb 1978.