ConAuth - context for authentication (Dec. 2017)

Saurabh Sharma, *saurabh.sharma@sv.cmu.edu*
Omar Serrano, *omar.serrano@sv.cmu.edu*

***Abstract*—With the growing number of wireless devices, we need efficient mechanisms to let the wireless devices communicate securely. The wireless devices sometimes share common sensors that can be leveraged to perform additional authentication procedures on a set of localized wireless devices. The problem which prevents such a judicious use of sensors is the orientation of wireless devices. Sensors such as gyroscope and accelerometer are commonly found in wireless devices, but their readings make no sense until their orientations are the same. We plan to conduct controlled experiments to investigate how different environmental factors impact the accelerometer performance and how the best accuracy can be achieved in an appropriate condition range. Based on such comprehensive understanding, we propose to develop a proof of concept which can be used to make a simple challenge-response authentication protocol between a car and the mobile device, using opportunistic calibration of the accelerometer.**

***Index Terms*—Contextual security, sensor fusion, Madgwick, device orientation, deep learning, keras, dropout, convolutional neural network, cross-validation, time synchronization, RTCDue, MPU9250.**

I. Introduction

With the growing number of IoT devices, securely pairing a new device into an existing set of devices is an extremely important yet burdensome task. Traditionally, these devices are paired manually, where an operator sets up an authentication with the existing network of devices. Specifically, we address the problem of a platoon ghost attack wherein an attacker device spoofs presence within a platoon to gain admission and subsequently execute malicious attacks [1]. To address such concerns, we explore the notion of fingerprinting device sensor readings for a device's context.

Devices that share context are expected to experience similar events. For example, two magnetometers in proximity are likely to process similar events if a magnetic strip is drawn close to them. Even if the readings are not exactly the same, the devices are likely to exhibit similar patterns as a result of the magnetic disturbance caused by the magnetic strip. Two video cameras, despite having different points of view, might be able to determine that they share context on the basis that both of them detect an object with similar shape and color; for example, if one of the video cameras records an individual with a blue shirt walking toward it, and the other camera records an individual with a blue shirt walking away from it.

There is a wide range of possibilities in how devices and sensors are used to determine context from a wide variety of physical stimuli, or how the scale of context is defined (e.g., school building vs a single room); however, we limit our research to the small context of a car, and to 3-dimensional orientation, acceleration, and magnetic sensor readings. Instead of relying on traditional methods (e.g, Kalman filter) to determine context from the type of sensors we are using, we employ a convolutional neural network, a popular technique for solving computer vison problems, to create a predictive model.

The problem that we are trying to solve is one of device pairing in an automobile context. For example, if you want to connect your mobile device to the Uber car that you are using for your commute, you want an effortless way to pair your device to the car. This process of pairing has to be secure from the driver's and customer's point of view. So an example use-case can be, the new version of Uber app gives it's customer the luxury of listening to their own songs while driving in an Uber. Uber does not want it's driver's to carry out the burdensome task of pairing themselves, so in their application, they have introduced a context-aware authentication protocol that lets the application running on customer's mobile phone to get the fingerprints of the context observed by the car and compares it to the fingerprint of context seen by the mobile. Based on the comparison the application makes a decision whether the customer can pair his phone to the car or not. This is just a simple application of the problem we aim to solve. There can be a plethora of applications of context-aware authentication.

II. Related Work

In this paper, we have come up with a proof of concept that helps in determining the context of a device with respect to other device. The problem that we aimed to solve was to make the sensor data of the two devices comparable using sensor reorientation.

The initial motivation of solving the problem of sensor reorientation was proposed in the paper: Design, Implementation and Evaluation of a Smartphone Position Discovery Service for Accurate Context Sensing. This paper talks about the difficulties that prevented the the large-scale proliferation of context-aware applications in the market. A major barrier for it was poor accuracy [2]. We address one of the key reasons for this poor accuracy, which is the impact of sensor orientation.

Devices have their sensors oriented in different positions. The paper first shows that smartphone positions significantly affect the values of the sensor data being collected by a context-aware application, and this in turn has a significant impact on the accuracy of the application [2]. Next, it describes the design and prototype development of an orientation discovery service that accurately detects a sensor orientation. This service is based on the sensor data collected from carefully chosen sensors. Finally, the paper demonstrates that the accuracy of an existing context-aware service or application is significantly enhanced when running in conjunction with the proposed orientation discovery service.

The motivation of our work was from Convoy: Physical context verification for vehicle platoon [1]. This paper developed an authentication protocol for a vehicle platoon where the convoy formed a network and shared continuous contextual fingerprints between the master and other participants to give a confidence score used in authentication of the members of the network. This protocol explicitly prevented platoon ghost attack, wherein an attacker spoofs presence within a platoon to gain admission and subsequently execute malicious attacks.

The other important work that we relied on was Use It Free: Instantly Knowing Your Phone Attitude [3]. This paper helped us get a brief idea of the nuances involved in sensor orientation. The logic that we finally applied for sensor reorientation was derived using the concepts learned in this paper. This paper talks about identifying your phone's attitude using gyroscope and accelerometer using Madgwick algorithm. The goal was to find the orientation of the device with respect to Earth's frame of reference.

Sensor orientation information is provided by euler angles and quaternions. Euler angles are simple and intuitive, and they are simple for analysis and control. But, we do not use Euler angles for getting the sensor orientation because they are limited by a phenomenon called gimbal lock, which prevents them from measuring orientation when the pitch angle approaches +/- 90 degrees. We, therefore, use quaternions, which is a four-element vector that can be used to encode any rotation in a 3D coordinate system. To use the Madgwick's algorithm to find the euler angles and quaternion representation of the sensor data, we used a open source Matlab library [4], which we further modified for sensor reorientation and displaying graphs of before and after sensor reorientation.

To complete the work for this paper, we relied on multiple open source libraries to collect data, and to build a neural network architecture to generate a model for predicting the context of a device. We were able to use the MPU9250 SparkFun library [5], with minor modifications, to collect three-dimensional orientation, acceleration, and magnetic sensor readings from the PowerDue. To be able to compare the data from the PowerDue and the mobile phone, readings from both devices had to be time-stamped, and the RTCDue [6] and Time [7] Arduino libraries allowed us to generate timestamps with the PowerDue. Implementing neural network architectures is not a trivial task, but Keras [8], a python deep learning library which uses TensorFlow as a backend, made it easy for us to create a neural net that we could then train and use to make predictions. While Keras made it easy to implement a neural network, [9], [10] inspired us to try a deep learning solution, and [10], [11] were helpful in understanding the intricacies of using neural networks effectively.

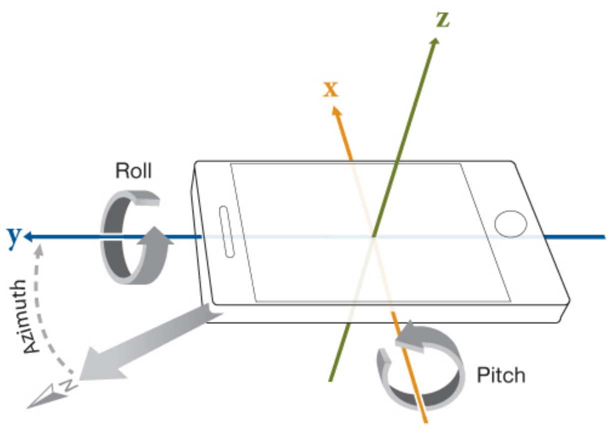


Fig. 1. A device's body orientation.

III. Approach

To build a model to determine whether two devices share the same context, we took the following approach:

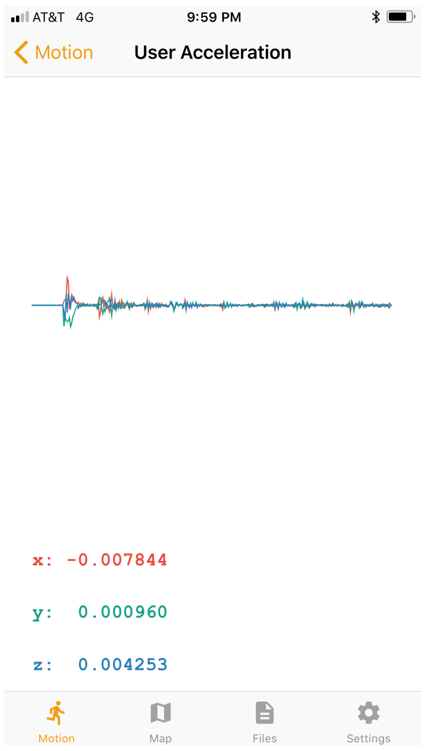
1) Modify the MPU9250 library to work with the PowerDue.
2) Establish a clock on the PowerDue, to allow us to synchronize readings from the MPU9250 with readings from PowerSense.
3) Apply the Madgwick algorithm.
4) Run three experiments to collect data from the MPU9250 and PowerSense iOS app. See Figure 2 and Figure 3 for an example of the type of data collected with PowerSense.
5) Use the unix timestamps to aggregate MPU9250 and PowerSense data.
6) Normalize the data.
7) Build a neural network architecture.
8) Train the neural network.
9) Evaluate the model.

A. Modifying MPU9250 library

The MPU9250 library, which implements an interface to interact with the MPU9250 sensor, which contains a 3D gyroscope, accelerometer, and magnetometer, required only one modification to work with the PowerDue, updating the instances of the `Wire` interface with `Wire1`, given that PowerDue has two wire interfaces. In addition to this modification, we used a library example module [12] as the basis for the PowerDue module that we used to obtain readings from the MPU9250 sensor. We disabled most of the logic in the example module, but relied on critical sections of the module, such as initializing and calibrating the sensor.

B. Time Synchronization

To be able to compare the readings from both devices, each reading had to be timestamped, and the timestamps from the devices had to be relatable, i.e., one timestamp could be converted to the other. PowerSense readings provided a unix timestamp by default, and this was ideal, but the PowerDue

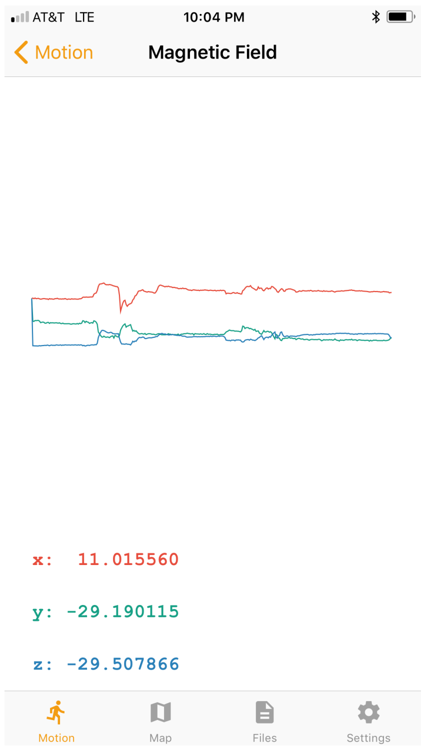


x: -0.007844

y: 0.000960

z: 0.004253

Fig. 2. User acceleration with iOS app PowerSense.



x: 11.015560

y: -29.190115

z: -29.507866

Fig. 3. Magnetic field with iOS app PowerSense.

did not not have a system clock established, i.e., calling a clock time function would begin counting time from zero.

To establish the system clock on the PowerDue, we used RTCDue, a library that allows you to set the time in different ways [6], e.g., using a unixtime stamp or a date. Ideally, we would have preferred to use the NTP protocol to set the time on the PowerDue, because the time would have a greater degree of accuracy. Unfortunately, we were unable to use NTP because we were unable to get the Wi-Fi module to work. We tried approaches in different example modules found within the PowerDue libraries, but none of them worked. Therefore, we resorted to a compile-time mechanism for setting the time.

To set the time at compile-time, we passed in the macros `__TIME__` and `__DATE__` to a couple of RTCDue library functions to set the time and date. Essentially, we used the compiler to pass in the time when it built the binary. Simply setting the time like this did not yield a very accurate time, because of the delay between macro substitution at compile-time and program execution. To offset this delay, we reset the time at runtime by adding seconds to the time set with `__TIME__` and `__DATE__`, essentially using inspection to pick the number of seconds that made the time reported by PowerDue closer to the time reported by our machines and PowerSense.

After setting the system clock, it was simply a matter of using the Time library function `now` [7] to output the unix timestamp with the readings from the MPU9250. Ideally, we would have preferred a millisecond accuracy level, which we might have obtained using NTP, but the compile-time method of setting the time provided accuracy of within a second, allowing us to carry on with the experiments.

C. Madgwick algorithm

Madgwick provided a novel orientation filter applicable to tri-axis gyroscopes and accelerometers. The filter uses quaternion representation instead of Euler angles to prevent gimbal lock phenomenon. For sensor reorientation, we had to to use the open source madgwick algorithm implementation to find the quaternion representation of the two data sets. The methodology is explained in the following steps:

- Import the data from a .mat file format and plot the sensor data.
- Process sensor data through the algorithm to find the quaternion representation.
- Use this quaternion representation to find the relative orientation.
- Reorient the sensor data with respect to the relative orientation.

The basic logic of calculating the relative orientation was to divide the quaternion representations with respect to each other, shown in eq.1. In quaternion representation, you can also perform the multiplication of inverse in place of division, shown in eq.2. An inverse of a quaternion is also called its conjugate. So in eq.3, you can see the conjugate of $quaternion1$ is multiplied to $quaternion$, because we want to find the orientation of $quaternion1$ with respect to $quaternion$. In eq.4 we get the reoriented quaternion representation of $quaternion1$.

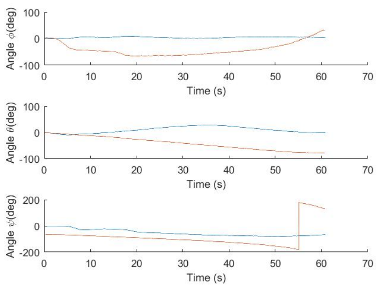


Fig. 4. Euler Angles before reorientation.

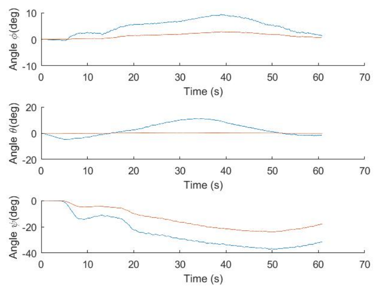


Fig. 5. Euler Angles after reorientation.

$$relative_o r = (quaternion_1 / relative_o r) \tag{1}$$
$$relative_o r = (inverse(quaternion_1) * relative_o r) \tag{2}$$
$$relative_o r = quaternion_1 .* quaternion \tag{3}$$
$$quaternion_1 = (quaternion_1 .* relative_o r) \tag{4}$$
$$euler = quaternion * (180/pi) \tag{5}$$

Quaternions cannot be represented as a graph. Therefore, the quaternions are converted into euler angles using eq.5, just to show the graphical representation of sensor reoriented data.

Figure 4 gives the graphical representation of the yaw, pitch and roll orientation in terms of euler angles before applying sensor reorientation.

Figure 5 gives the graphical representation of the yaw, pitch and roll orientation in terms of euler angles after applying sensor reorientation.

D. Experiments

1) Apparatus

1) Honda Accord
2) Ford Lincoln
3) iPhone with PowerSense
4) Nexus 5 with ArduinoSense
5) PowerDue board
6) Timer

2) Methodology

Our motivation for designing the experiments was to test different contextual environment between two devices. We conducted three sets of experiments from which we collected data for five iterations for each set of experiment. Each iteration of the experiment was for 90 seconds. The reason for selecting three experiments was to introduce a clear pass case ($shared context$), clear fail case ($noshared context$) and a case where the context is shared but not at the same point in time ($Time - based context sharing$). The third experiment is very crucial from a security point-of-view, because it demonstrated the case of an adversary who is following your car in the same lane and sharing the same context, but since this shared context is not time-based, we differentiate it from the context of the victim. We defined shared context to mean that both devices, the PowerDue, and the mobile phone, were inside of the same car. The experiments were:

- *Shared context*. Drive vehicle with both devices inside of vechicle.
- *No shared context*. Drive vehicle with PowerDue inside vehicle, while mobile phone is carried by pedestrian.
- *Time-based context sharing*. Drive vehicle with PowerDue inside vehicle, and drive another vehicle, following the first vehicle with mobile phone inside it.

An essential ingredient of all the experiments was movement, in order to provide stimulus for the sensors, and hence our decision to carry the experiments while driving or walking. The output of each experiment was a set of data files. PowerSense allowed us to email the readings of an experiment as a CSV file to ourselves. For PowerDue, we had to manually create a CSV file by copying the output from the serial console to the file.

E. Aggregating data

There are a number of reasons why we aggregated the data from two data files to one data file per experiment. One is convenience (e.g., uploading one data file as a data frame vs multiple data files), but more importantly, to match the readings from each device, which we did in a three step process.

1) Find the timestamps (i.e., seconds) where both devices output readings.
2) From each device's data file, select the readings with a timestamp from step 1.
3) In cases where the number of readings for a given second don't match, apply a transformation to make the number of readings for that second equal for both devices.

Even though we tried to start and stop reading from each device at the same time, it is inevitable to have slight variation when taking samples from more devices, especially if there is a human element to the experiment, which there was in our experiments because we started the readings manually by flashing the program on the PowerDue and pressing a button on the PowerSense app. Therefore, we used only readings with timestamps existing in both data files, per step 2.

In step 3, we had to apply some transformations because PowerSense produced 50 readings per second, while PowerDue only produced 38 readings per second. To smooth-out this difference, we mapped pairs of readings from PowerSense into a single reading with the average of both readings, but we applied this only to the tail end of the readings from PowerSense. For example, suppose PD_s is the set of readings

$a_1, a_2, ..., a_{38}$ from the PowerDue, and PS_s is the set of readings $b_1, b_2, ..., b_{50}$ from PowerSense, for second s, and a_i and b_i represent vectors. Then the transformation applied to PS_s would map the last 24 readings to 12 readings, as indicated below:

$$b'_{26} = \frac{b_{26} + b_{27}}{2} \quad (6)$$

$$b'_{27} = \frac{b_{28} + b_{29}}{2} \quad (7)$$

$$\vdots \quad (8)$$

$$b'_{38} = \frac{b_{49} + b_{50}}{2} \quad (9)$$

F. Data normalization

After aggregating data by time, we normalized the data in two ways:

- Normalized all the data values (i.e., x, y, z and values for orientation, acceleration, and magnetic), so that the mean would be zero and the standard deviation 1.
- Made all input arrays, obtained from each aggregate experiment file, have the same dimensions.

We normalized all the data values because having training data with a mean of zero tends to speed up convergence [13]. We also normalized every input array to have the same number of rows, in order to be able to use a neural network architecture with an convolutional layer and a dense output layer. To make all the arrays have the same number of rows, we computed the average number of rows per array (15 total arrays, one per experiment trial), and removed rows from the end of arrays with more rows, or repeated the last row at the end of arrays with less rows.

G. Neural network architecture

Our neural network architecture consists of 5 layers:

1) Input convolutional layer with a relu activation function.
2) Hidden convolutional layer with a relu activation function.
3) Hidden max pooling layer.
4) Hidden dense layer with a relu activation function.
5) Output dense layer with softmax activation function.

With Keras [8], it was trivial to implement the network, as the following lines of python code demonstrates

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3),
                 activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))
```

Even though the data we collected is inherently time-dependent, the way our experiments were carried makes time less of an important variable, because we did not carry out any experiments in which we tested changing context, i.e., going from sharing context to not sharing context. Such an experiment would have required us to look at how the variables change over time, precisely so that we could devise a model that can distinguish the moment when a device's state changes from being in context to not being in context.

In our experiments, the devices are either in context, or not in context, there is no change from one to the other. This motivated our decision to use a convolutional neural network, where time is not a factor. We simply wanted to classify the input as either in context, or not in context, hence an output layer using a softmax function [9], [10].

From the code, it may seem like the network also has a `Dropout` layer; however, dropout is not inherently part of the network, but rather a technique used to randomly drop units of the network during training in order to prevent overfitting of the model [11].

H. Training the network

Given the small amount of data at our disposal, we used k-fold cross-validation with $k = 15$, or leave-one-out cross-validation [14]. To use this approach, we created and trained a model with 15 different sets of training data, each time evaluating the model on a single instance, the one left out, of the data.

I. Evaluating results

Overall, the model's prediction rate is somewhat surprising, because it predicts 8 out of 15 instances correctly, despite the small amount of data used to train the models. However, on closer inspection, the results do not inspire a lot of confidence on the model's prediction effectiveness, given that it predicts almost all of the samples, except two, as not sharing context, and in both cases where it predicts that the devices are sharing context it does so incorrectly.

Figure 6 illustrates the prediction results. The graph on the left contains the predicted values, the graph on the right contains the actual values, 1 and 0 represent context shared or not shared in the vertical axis, and the horizontal axis represents the experiment trial. Thus, we can see that the models predicts that data for trials one through five, where the context is shared, as context not being shared. Therefore, and it may be a side effect of the fact that we have more samples of data where the devices are not sharing context, that the model is biased toward making predictions for context not shared.

IV. Conclusion

There are many flaws in the work conducted for this experiment, including time synchronization issues, the lack of a system for automating the process of collecting experimental data, having only a small amount of data to train our models, creating a model that does not take into account the fact that the state of context being shared is dynamic.

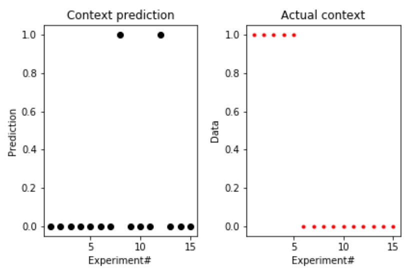


Fig. 6. Model predictions vs actual data for devices sharing or not sharing context.

Time is a key ingredient of whether two devices share context or not, and hence time should be synchronized down to the level of milliseconds. Using NTP would have made our results more accurate, but it would be even better to develop a communication procotol that allows two devices to quickly reach consesus on a measure of synchronicity, whether by time, a nonce, or some kind of counter. Such a protocol does not need to be tied to hardware or sensors, and hence could be applied anytime devices are trying to synchronize with each other.

We only collected 15 samples of data because it was very time consuming to collect them - it took us about 3 hours to collect 15 samples. In addition to being time-consuming, it was error prone, given that most of it was done manually. In one case, we forgot to copy and paste the output from one of the experiments before proceeding to run the next experiment. Automating this process, for example, by having PowerDue collect samples for some time before sending them to the cloud, or an edge server, would allow us to collect more data in a more effective and accurate manner.

Devices that are bound to a location have a static shared context with other devices that are also bound near by; however, this is not a very interesting instance of devices sharing context, because such devices can be programmed to recognize that they share context. What's interesting about context being shared between devices, or not, is that it is more likely to be dynamic, with devices moving within and out of a state of shared context. Even devices that are bound to a location will experience a change of state, because other mobile devices may enter or leave their vicinity. Therefore, it would be more interesting and realistic to develop a model that takes this dynamic into account, and hence it would be more appropriate to use a different kind of neural network, such as an LSTM network, or to simply use a different technique that is more appropriate for time series and modeling state. To test such a model, it would be necessary to devise experiments where devices change from one state to the other.

References

[1] J. Han, M. Harishankar, X. Wang, A. J. Chung, and P. Tague, "Convoy: Physical context verification for vehicle platoon admission," in *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '17, DOI 10.1145/3032970.3032987, pp. 73–78. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3032970.3032987

[2] K. Alanezi and S. Mishra, "Design, implementation and evaluation of a smartphone position discovery service for accurate context sensing," *Comput. Electr. Eng.*, vol. 44, DOI 10.1016/j.compeleceng.2015.01.015, no. C, pp. 307–323, May. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.compeleceng.2015.01.015

[3] P. Zhou, M. Li, and G. Shen, "Use it free: Instantly knowing your phone attitude."

[4] [Online]. Available: https://github.com/arduino-libraries/MadgwickAHRS

[5] [Online]. Available: https://github.com/sparkfun/SparkFun_MPU-9250_Breakout_Arduino_Library

[6] [Online]. Available: https://github.com/MarkusLange/RTCDue/

[7] [Online]. Available: https://playground.arduino.cc/Code/Time

[8] [Online]. Available: https://keras.io/

[9] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, DOI 10.1038/nature14539, no. 7553, pp. 436–444, 5 2015. [Online]. Available: http://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

[12] [Online]. Available: https://github.com/sparkfun/SparkFun_MPU-9250_Breakout_Arduino_Library/blob/113e836e989699913660c60bacda12da4a919c91/examples/MPU9250BasicAHRS_I2C/MPU9250BasicAHRS_I2C.ino

[13] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pp. 9–50. London, UK, UK: Springer-Verlag, 1998. [Online]. Available: http://dl.acm.org/citation.cfm?id=645754.668382

[14] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

Appendix

Work completed by Omar Serrano: milestone update report and presentation; modification to MPU9250 library; time synchronization with RTCDue and Time libraries; script for aggregating data, convert_raw_data.py; IPython notebook to build and train neural network model; GitHub repo readme; final project report and presentation; conduct experiments; literature review.

Work completed by Saurabh Sharma: milestone update report and presentation; final project report and presentation; madgwick script, Example2.m, modified in matlab; reorientation of quaternions, literature review, conduct experiments.