Gesture Recognition using Sensor Fusion & Deep Learning on an Arduino Embedded Platform
EE292D Course Project

David Whisler
Stanford University
dwhisler@stanford.edu

1. Introduction

Computers have changed and improved drastically since their invention, however, the way humans interact with them has remained at its core the same. The traditional mouse and keyboard has been the preferred method of interface for many years. Even since the large-scale adoption of touch screen technology, the fundamental interaction method is still largely with a virtual keyboard. Gesture-based interaction methods have potential as a more natural method for human interaction with traditional computers, and in addition there are many potential applications within the growing field of virtual and augmented reality systems.

Gesture recognition can be implemented using vision, however this can be tricky to implement due to dependence on line-of-sight and lighting conditions, and is awkward to use in handheld devices. Nearly all smartphones and devices such as smartwatches have built in accelerometers or full IMU sensors that can provide high-resolution, high-bandwidth motion data utilizing advances in MEMS technology. This data is already widely used in sensing basic orientation, but it can be difficult to perform sensor fusion and to extract more complex data. Recent advances in machine learning pose an opportunity for more complex gesture recognition from raw sensor data, and quantization allows for trained models to run in resource-constrained embedded environments. This new opportunity in embedded machine learning enables improved gesture recognition on mobile devices and even more constrained devices such as microcontrollers attached to gloves or fingertips.

2. Related Work

Prior work in this space has explored using IMU data in smartphones and smartwatches for human activity recognition (HAR) [4]. This involves determining an activity state, such as running, walking, jogging, and sitting. Most HAR models use recurrent neural networks (RNNs) consisting of long-short-term-memory (LSTM) units, which are naturally suited to sequence data. In addition, this model used accelerometer data alone. Another reference for gesture recognition is the Magic Wand example in the Tensorflow Lite examples code, also featured in the book TinyML [2]. This approach uses a convolutional neural network (CNN) to classify gesture data, also consisting of only accelerometer data.

3. Methods and Approach

The sensor fusion algorithm and neural network model were implemented on an Arduino Nano 33 BLE, which has a built-in 9-axis IMU (LSM9DS3). In order to train a deep learning model to recognize gestures, data from the IMU was used. One goal of this project was to determine if pre-preprocessing the raw IMU data into a relative orientation using a sensor fusion algorithm and training on the transformed data would outperform training on the raw IMU data without the sensor fusion. By doing so, it can be demonstrated if the end-to-end approach is superior for gesture recognition, letting the model "learn" the sensor fusion on its own, of if it is superior to use domain knowledge in the form of the sensor fusion algorithm to preprocess the data and train on that.

3.1. IMU Data

The IMU sensor used has 9 axes of available data, including a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetometer. However, because of the sensitivity to magnetic distortions in the environment, the difficulty in calibration, and because relative orientation is more important than absolute orientation in gesture recognition, the magnetometer data was not used.

3.2. Sensor Fusion

In general, accelerometer data tends to be noisy, having a high variance. By contrast, gyroscope data has a lower variance, but it has the tendency to drift over time, which

is exacerbated by the need for integration of angular velocity to determine orientation. By combining orientation estimates from both sensors, we can achieve the "best of both worlds" with a low variance, bias-corrected estimate of relative orientation. This consists of using a complementary filter, which takes a weighted sum of orientation predicted by the accelerometer and orientation predicted by the gyroscope. The derivation for the sensor fusion algorithm is based on course notes from the Stanford class EE267 by Gordon Wetzstein [1].

3.2.1 One Dimensional Case

To demonstrate, first consider a flat environment, predicting a single-dimensional orientation angle. An orientation estimate is calculated by integrating the angular velocity estimates on each axis from the gyroscope in a first-order Taylor approximation, as below.

$$\theta(t+\Delta t) \approx \theta(t) + \frac{\partial}{\partial t}\theta(t)\Delta t \tag{1}$$

$$\theta_{gyro}^{(t)} \approx \theta^{(t-1)} + \omega^{(t)}\Delta t \tag{2}$$

Then, using the assumption that the acceleration vector always points up on average, an orientation estimate using the accelerometer data can be found by

$$\theta_{acc}^{(t)} \approx \text{atan2}(a_x^{(t)}, a_y^{(t)}) \tag{3}$$

Then, these estimates are combined using a weighted sum in the complementary filter, for some $\alpha \in [0, 1]$.

$$\theta^{(t)} \approx \alpha\theta_{gyro}^{(t)} + (1-\alpha)\theta_{acc}^{(t)} \tag{4}$$

3.2.2 Three Dimensional Case

In the three dimensional case, it is advantageous to represent orientation not as Euler angles (roll, pitch, and yaw), but as a 4-dimensional quaternion. Quaternions have the advantage of avoiding the problem of "gimbal lock", where the orientation cannot be uniquely represented using Euler angles when one or more coordinate axes align and causes numerical instability. A quaternion can be defined in terms of a rotation axis v and rotation angle θ about that axis, as shown below.

$$q(\theta, v) = \cos\left(\frac{\theta}{2}\right) + iv_x \sin\left(\frac{\theta}{2}\right) + jv_y \sin\left(\frac{\theta}{2}\right) + kv_z \sin\left(\frac{\theta}{2}\right) \tag{5}$$

where i, j, and k are imaginary units such that $i^2 = j^2 = k^2 = ijk = -1$. Quaternions have multiplication, conjugate, and inverse rules, which can be seen in the reference [1].

The first step is to approximate the orientation using the gyroscope data and integrating with the first order Taylor expansion, as in the one-dimensional case. This quaternion is defined in the angle-axis formulation.

$$q_\Delta = q\left(\Delta t||w||, \frac{w}{||w||}\right) \tag{6}$$

$$q_{gyro}^{(t)} \approx q^{(t-1)} q_\Delta \tag{7}$$

Then, a quaternion is calculated from the acclerometer data, allowing for tilt correction under the same assumption that the acceleration vector on average points up (in the y direction) with gravity. The quaternion is again constructed from the angle-axis formulation, with a normalized rotation axis n and a rotation ϕ.

$$n = \frac{1}{||a||}\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \frac{1}{||a||}\begin{bmatrix} -a_z \\ 0 \\ a_x \end{bmatrix} \tag{8}$$

$$\phi = \cos^{-1}\left(\frac{a_y}{||a||}\right) \tag{9}$$

$$q_{acc}^{(t)} = q\left(\phi, \frac{n}{||n||}\right) \tag{10}$$

Then, the two estimates are combined using the complementary filter, based on the parameter $\alpha \in [0, 1]$.

$$q^{(t)} = q\left((1-\alpha)\phi, \frac{n}{||n||}\right) q_{gyro}^{(t)} \tag{11}$$

3.3. Bias Correction

To account for bias in the gyroscope and accelerometer data, upon sensor initialization the average value for each of the acclerometer and gyroscope axes is found, and this bias is subtracted at each time step before using the values in orientation calculations.

$$a_{bias} = \frac{1}{N}\sum_{i=1}^{N} a_i \tag{12}$$

$$w_{bias} = \frac{1}{N}\sum_{i=1}^{N} w_i \tag{13}$$

$$a^{(t)} = \tilde{a}^{(t)} - a_{bias} \tag{14}$$

$$w^{(t)} = \tilde{w}^{(t)} - w_{bias} \tag{15}$$

To make the model more robust to whole-body movements, and a better representation of relative orientation for gestures (so that if a hand is tilted slightly in different orientations, the same gestures will still be recognized), an additional method was used for bias correction. This involved updating the bias with an exponential moving average, which in effect served as a low-pass filter that mitigated the effect of whole-body movements.

$$a_{bias}^{(t)} = \beta\tilde{a}^{(t)} + (1-\beta)a_{bias}^{(t-1)} \tag{16}$$

Then, depending on the parameter β, the bias term would approach the average value of recent accelerometer measurements, which makes the model more robust to large movements. The strength of this moving average depends on β; for example, if $\beta = 0.01$, then the exponential moving average would effectively average the last $1/\beta$ or 100 samples.

3.4. CNN Model

To learn a function to map input data to gestures, a convolutional neural network (CNN) was used as a classifier. Normally, recurrent neural networks (RNNs) tend to be better suited to classify sequence data such as IMU data, however, we are limited by the embedded environment. Tensorflow Lite Micro was used to deploy the trained model to the Arduino, which as of June 2020 does not have as much support for operations and models like RNNs and LSTM (long short-term memory) units. Instead, a CNN model was used, based on the Magic Wand [2] gesture recognition model from the book TinyML. This model was defined in Keras as

Keras Layer Definition

```
Conv2D(8,(4, d),padding='same',activation='relu',input_shape=(s, d, 1))
MaxPool2D((3,d))
Dropout(0.1)
Conv2D(16, (4, 1),padding='same',activation='relu')
Dropout(0.1)
Flatten()
Dense(16,activation='relu')
Dropout(0.1)
Dense(c,activation='softmax')
```

where d is the number of data axes, s is the sequence length, and c is the number of gesture classes.

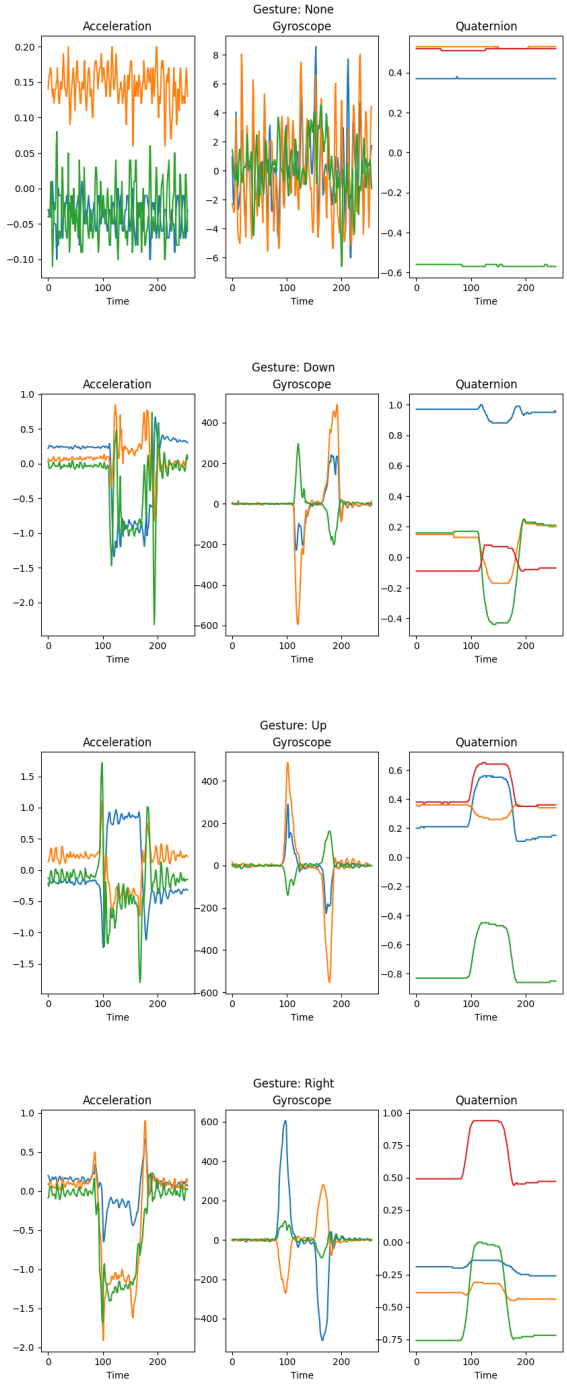
3.5. Gesture Dataset

For the model, four gesture classes were used. With the assumption that the Arduino was attached to the index finger of the right hand, this consisted of "none" (no gesture), "down" (a downward rotation of the wrist, fingers outstretched), "up" (an upward rotation of the wrist, fingers outstretched), and "right" (a rightward twist of the wrist, fingers outstretched). 50 training examples were taken for each class, with a sequence length of 256 samples, which translated to approximately 2 seconds. For each sample, 10 data axes were recorded - the 3 raw accelerometer axes, the 3 raw gyroscope axes, and the 4 sensor fused quaternion orientation values.

This gave each data point fed into the model an input shape of 256xdx1, where d varied according to the experiment and represents the number of data axes used in the model. Examples of each data class are shown below, plotting the accelerometer data, gyroscope data, and sensor fused quaternion orientation data.

3.6. Data Augmentation

To augment the collected dataset, for each of the 50 training example, a circularly shifted version of the 256 sample example was also included with the same label, effectively



multiplying the number of examples for each class by 256. This yielded a total of 12,800 training examples per class. This was appropriate because in the inference stage, data is continuously fed into a circular buffer on which the inference is run. Since there is no way to guarantee that the gesture align a certain way with the circular buffer, nor should it be aligned, training on each circularly shifted permutation would enhance the robustness of the model.

3.7. Model Quantization

After training in Tensorflow, the model was then quantized and converted to a TFLite Flatbuffer for deployment on the Arduino. This was done with full 8-bit integer quantization, except for the inputs and the outputs.

4. Results & Analysis

For training each model, a train-test split of 80-20 was used. A batch size of 4 was used, and the Adam optimizer was used to train with a learning rate of 1e-3. Each model was trained for 2 epochs.

4.1. Input Data Type Comparison

The first set of experiments sought to determine what effect the preprocessing sensor fusion had on the accuracy of the model. Five models with different input data were tested, using the input data as defined below:

1. Raw accelerometer data only ($d = 3$)
2. Raw gyroscope data only ($d = 3$)
3. Raw accelerometer and gyroscope data ($d = 6$)
4. Sensor fused quaternion orientation only ($d = 4$)
5. Raw accelerometer, raw gyroscope, and sensor fused quaternion orientation ($d = 10$)

To evaluate each method, the accuracy of the test set during training time was found. In addition, an experiment was performed at runtime on the Arduino where 16 gestures were performed (4 for each class), and the prediction accuracy was recorded.

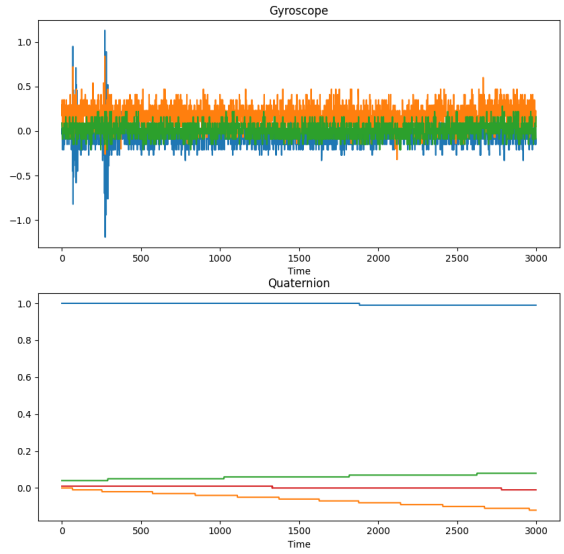
Model	Training Accuracy	Test Accuracy
1	1.0000	1.0000
2	0.9837	1.0000
3	0.9865	1.0000
4	0.9986	1.0000
5	0.9783	1.0000

Training time results

Model	None	Up	Down	Right
1	4/4	4/4	4/4	4/4
2	4/4	0/4	4/4	2/4
3	4/4	1/4	1/4	1/4
4	1/4	1/4	3/4	0/4
5	4/4	0/4	0/4	1/4

Runtime results

As can be seen, at training time, all the models essentially trained perfectly, achieving 100% accuracy on the test set. However, at runtime on the Arduino, a clear difference can be seen in model performance, with the accelerometer-only model (#1) being the only one achieving 100% accuracy. This can be attributed to sensor drift in the gyroscope, which is especially prone to this issue. All of the models 2-5 incorporate the gyroscope data in their input data, whether directly as raw data, or indirectly as sensor fused quaternion orientation. This skewed the data present at runtime versus the data at training time, causing the worse accuracy. The drift effect can be seen in plots of raw gyroscope data and sensor fused quaternion data over a longer period of time.



4.2. Bias Correction Method Comparison

After determining that the raw accelerometer data performed the best in classifying the gesture data at runtime on the Arduino, an experiment was performed to determine the effectiveness of the bias correction techniques. This bias correction helped to make the model more robust to only be sensitive to relative orientation, rejecting whole-body movements that could skew the intended gesture. The exponential moving average method of updating the accelerometer bias was used, as described in equation (16). The two models below were tested, comparing the accuracy of 16 performed gestures (4 per class) in 3 different hand orientations (left tilt, right tilt, and neutral tilt) for total of 48 trials per model.

1. Raw accelerometer data only with naive bias correction (equation 14)
2. Raw accelerometer data only with exponential moving average bias correction (equation 16)

Model	Training Accuracy	Test Accuracy
1	1.0000	1.0000
2	0.9985	1.0000

Training time results

Model	None	Up	Down	Right
1	4/12	5/12	8/12	10/12
2	12/12	8/12	9/12	12/12

Runtime results

As can be seen in the results, the exponential moving average bias correction made the model much more robust to large-scale changes in hand position, focusing instead on relative orientation.

4.3. Video Control Application

After creating a gesture recognition model with high accuracy, the output was used in an application to control a host computer. The recognized gesture was transmitted over a serial port to a host computer, where a Python script interpreted the recognized gesture to perform an action to control a YouTube video playback. YouTube provides keyboard shortcuts for various video actions, such as pause (`k`), rewind 10 seconds (`j`), fast-forwarding 10 seconds (`l`), and various others [3]. The Python script reads the incoming recognized gesture from the Arduino, and simulates a key press following the above mapping to pause, fast-forward, and rewind a YouTube video.

5. Conclusion

In conclusion, the results show that using domain knowledge to perform sensor fusion in advance of training may be useful in theory, in practice, it has limited usefulness because of the sensitivity to gyroscopic drift. If a better drift-compensation method or more accurate hardware is used, it still may be a useful tool. However, the accelerometer-only models performed close to perfect accuracy, and with the exponential moving average bias correction method, it can be made robust to large-scale changes in orientation which was another goal of using sensor fusion. For future work, it would be interesting to determine if there are certain gestures which are more easily classified with sensor-fused orientation versus with accelerometer data alone.

A. GitHub Repository

All code can be found at the following repository: `https://github.com/dwhisler/gesture_recognition`

References

[1] G. Wetzstein. EE 267 Virtual Reality Course Notes: 3-DOF Orientation Tracking with IMUs. *Stanford University*. 2020. `https://stanford.edu/class/ee267/notes/ee267_notes_imu.pdf`

[2] P. Warden & D. Situnayake. TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. O'Reilly Media Inc, 2020. `https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/micro/examples/magic_wand`

[3] Keyboard Shortcuts for YouTube `https://support.google.com/youtube/answer/7631406?hl=en`

[4] P. Agarwal and M. Alam A Lightweight Deep Learning Model for Human Activity Recognition on Edge Devices. `https://arxiv.org/pdf/1909.12917.pdf`