Penn Project Report
Machine Learning in Real Time Character Recognition

Abstract

This paper presents a MEMS based digital pen for handwriting recognition. While there are several devices which are capable of digitizing handwritten notes, all commercially available devices require secondary hardware to function. We propose a single device solution for digitizing text as it is written on a page. By calculating the dynamics of the pen from IMU measurements, we are able to estimate the pen tip accelerations and rotations as a function of time. We evaluated SVCs, decision trees, Ada-Boost, and logistic regression learning algorithms based on their ability to recognize handwritten characters and achieved 97.9 % accuracy using an SVC with a lineal kernel.

1. Prototype

In this section we give a brief summary of the hardware incorporated into the proposed device. We first describe the mechanical device before then discussing the electronics.

1.1. Hardware

The pen layout and body was designed in CAD (pictured below) to ensure appropriate packing of components. The clear body was 3D printed in ABS in the University's prototyping labs.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

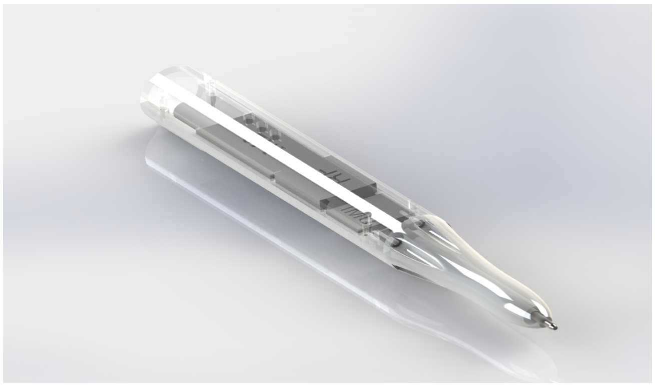

Figure 1. Rendering of the 3D printed model of the penn

1.2. Electronics

An ATMega32u4 microcontroller provides on-board processing and interfaces to the wireless module, IMU, switch, and indicator LEDs. Two Nordic nRF24LE1 transceivers handle wireless communication between the pen and base station computer. Inertial measurements are provided by an MPU6050 digital IMU with configurable gain triaxial accelerometers and gyroscopes. A Honeywell HMC5883L triaxial magnetometer provides a reference to earth's magnetic field for orientation. A switch behind the pen tip detects contact with the page for character segmentation. Three LEDs on the case display pen state. The pen also includes on-board batteries, power management, and an on-off switch to track idle and writing time of the pen, which translates in useful and un-useful data.

1.3. Filtering the data

The raw data from the IMU pose issues for learning algorithms. Sources of variance can include posture, handedness, roll angle of the pen, and sensor drift. To minimize sources of error, we compute the trajectory of the pen tip on the page to generate features in a four step process.

1. IMU measurements are scaled to engineering units and are low-pass filtered to remove high frequency noise from vibration.

2. We use Madgwick's open source Kalman filter and at-

titude estimator to find the orientation of the pen relative to the page.

3. We transform the IMU measurements into the ground frame relative to the page and double-integrate the acceleration to estimate the displacements of the pen tip.

4. Finally, the position and velocity estimates are high-pass filtered to remove drift.

From this IMU data processing we obtain the x,y,z position, velocity, and acceleration of the pen in the page frame as well as the euler angles and angular rates of the pen in the page frame at 300Hz sample rate.
Down below we can see how the user interface for testing looks like, and an example of the trajectory reconstruction of a ”g” letter. In the MATLAB plots we can see all the readings of the IMU in real time, the attitude estimation of the penn in 3D (Which corresponds to the real position of the penn with a very slight delay), and the real time trajectory reconstruction of the writings with other parameters.

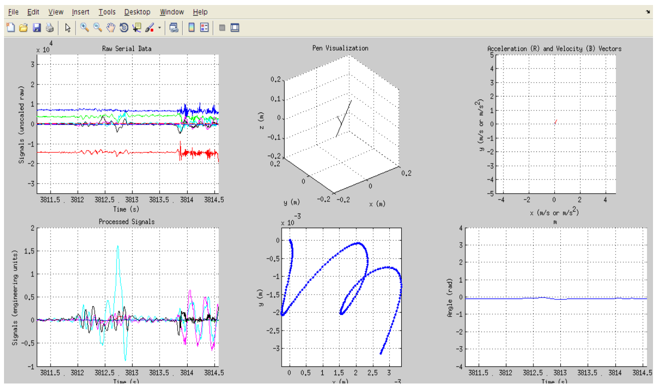
File Edit View Insert Tools Desktop Window Help
Raw Serial Data
Pen Visualization
Acceleration (R) and Velocity (B) Vectors
Processed Signals
Signals (unscaled raw)
Time (s)
Signals (engineering units)
z (m)
y (m)
x (m)
Angle (rad)

Figure 2. Writing interface for character recognition

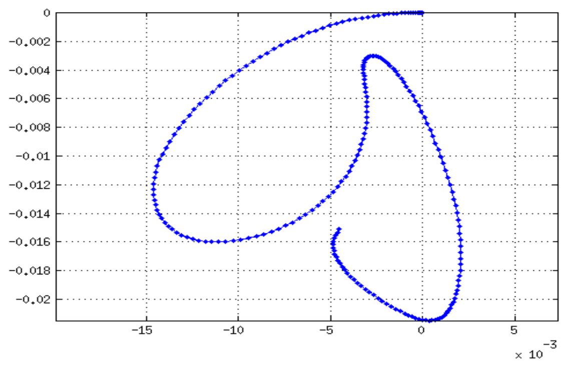

Figure 3. Recovered letter ”g”

1.4. Coordinate Transformation

Accelerations and rotations of the pen body measured at the IMU are translated to the tip by the following transform:

P: location of the IMU
O: origin of frame G on the page
frame G: SRT fixed to the page with origin O (ground frame)
frame M: SRT fixed to the pen at the IMU with origin at point P (pen frame)
Q: location of the pen tip
$\underset{\sim}{\mathbf{PQ}}$: vector from the IMU origin to the pen tip

The acceleration of the pen tip in the ground frame is given by:

$$^G\underset{\sim}{a}^Q = {}^G\underset{\sim}{a}^P + {}^M\underset{\sim}{a}^Q + {}^G\underset{\sim}{\alpha}^P \times \underset{\sim}{\mathrm{PQ}} + 2\,{}^G\underset{\sim}{w}^M \times {}^M\underset{\sim}{v}^Q + \ldots$$
$$+\,{}^G\underset{\sim}{w}^M \times \left({}^G\underset{\sim}{w}^M \times \underset{\sim}{\mathrm{PQ}}\right)$$

Since point Q (the pen tip) is fixed in frame M (the pen frame), ${}^M\underset{\sim}{a}^Q$ and ${}^M\underset{\sim}{v}^Q$ are both 0. The above equation simplifies to:

$$^G\underset{\sim}{a}^Q = {}^G\underset{\sim}{a}^P + {}^G\underset{\sim}{\alpha}^P \times \underset{\sim}{\mathrm{PQ}} + {}^G\underset{\sim}{w}^M \times \left({}^G\underset{\sim}{w}^M \times \underset{\sim}{\mathrm{PQ}}\right)$$

The IMU measures accelerations and angular velocities of frame M relative to frame G expressed in frame M. Using the attitude estimate, these measurements can be rotated into frame G to produce ${}^G\underset{\sim}{a}^P$ and ${}^G\underset{\sim}{w}^M$. The angular acceleration ${}^G\underset{\sim}{\alpha}^P$ is obtained by numerically differentiating ${}^G\underset{\sim}{w}^M$.

The trajectory of the pen tip in the page frame is obtained by twice-integrating ${}^G\underset{\sim}{a}^Q$. Since the velocity and position are integrations of noisy signals they are prone to very significant drift over time. To counteract this drift, the position and velocity estimates are re-zeroed at the start of every character and both the velocity and position are high pass filtered. The resulting trajectory can be plotted in the plane to approximate the appearance of the character on the page. This trajectory image is sampled to produce a black and white image which is then scaled and blurred to produce the final learning features.

2. Learning

2.1. Features

To generate our feature vector, we find the mean, standard deviation, and RMS of each of the signals x, y, z position, velocity, and acceleration and euler angles and angular rates, then interpolate each of the signals to a uniform length. Our feature vector consists of the character writing time, these interpolated signals, and their mean, standard deviation, and RMS. This generates a feature vector of length 15*N + 46 where N is the interpolated signal length. After feature vector generation we standardize our features.

2.2. Learning Algorithms

We first tested Support Vector Machines with different Kernels to compare the results we were getting from them. This ones were all runned with 5 folds.

Algo.	Kernel	param	Sig Len	Time(s)	Acc.
SVC	linear	-	10	2.114	96 %
SVC	poly	2	10	2.07	91 %
SVC	poly	3	100	6.681	91 %
SVC	gaussian	5E-1	10	5.057	3 %
SVC	gaussian	5E-4	10	2.124	77 %
SVC	gaussian	5E-5	10	3.05	29 %
SVC	gaussian	5E-4	100	11.357	93 %
SVC	gaussian	5E-5	100	13.709	82 %
SVC	gaussian	5E-3	1000	309.19	3 %

As we can clearly see, while the linear and polynomial kernel do a pretty good job to classify the characters, the gaussian is VERY prone to subtle variations in the parameters, which makes it a non robust method for character recognition. The gaussian kernel smoothes everything if you don't have the correct parameters, and that is why it's a non reliable method for this application in particular. We can see how the performance for the gaussian kernel goes from 3 to 77% and then back down to 10% depending on the choice of gamma. And then once again changing the signal length we get very different performances.
On the other side, we can see that the accuracy obtained by the polynomial kernel is not even affected by the degree of the same and the variations are little while it runs very fast. Better then, we get the best performance with the linear kernel, were we don't have to worry about the gamma or the polynomial degree.

As expected, decision trees perform poorly on this dataset. Decision trees in general have lower predictive power than the other methods applied here. Furthermore, we expect the ideal decision function to be much smoother non axis-aligned than the axes-parallel hyper-rectangles of decision trees can produce. This is because the integrated time series features are highly correlated with one another. Despite the smoothing of AdaBoost, AdaBoosted decision trees still underperformed other methods for some settings, and got over 97.9% accuracy for some other setting, but taking a lot of extra time that is essential for the quickness of writing and 3-4% increase in performance does not justify the speed of AdaBoost at this settings.

Algorithm	param	Acc.
DT	depth=100	67.82 %
DT	depth=200	68.24 %
Log Reg	L1 penalty	89.20 %
Log Reg	L2 penalty	89.47 %
AdaBoost	k=7 c=1	70.11%
AdaBoost	k=5 c=3	92.99 %

In this table we have k being the k-folds and c being cross validation splits. We also tuned the depth of the Decision Tree and the number of estimators and the learning rate for the algorithm. This could all tune the accuracy that we got from it, but also influenced a lot the time that the

Logistic regression performs well achieving accuracy, precision, and recall of 89% with both l1 and l2 norm distance metrics.

Several letters are very similar in stroke trajectory. All learning algorithms struggled with these sets of letters more than other sets of letters as the confusion matrices reveal. Similar letter pairs include o and 0, p and b, 9 and q, y and x, h and n, 0 and 6, 5 and 6, 7 and 8, v and r, m and k, and a and u. Some of these letter pairs look visually different from each other, but are composed of similar stroke patterns (for example a script k and an m share the same vertical beginning stroke, direction reversal, loop and second direction reversal).

3. Performance

3.1. Tested Algorithms

The algorithms that were tested were SVCs with different kernels, Decision Trees, Logistic Regression with L1 and L2 penalties, and AdaBoost decision Stumps. While all of the performed with a lot more than random success, there were differences between them.

3.2. Final Performance

As we can well see, even though the logistic regression gets 89% accuracy, the ones that do better are the SVCs with linear kernel. SVCs with linear kernels present a robust machine learning algorithm that is not going to be changed completely if we add a new feature to the feature vector for example. They perform quickly, efficiently, and accurately making them the best algorithm for the task. We will next compare some of the results between the SVCs with linear kernels and the Decision trees that among the ones that perform poorly, still get over 60% accuracy.

We can observe how the true positives in the SVCs go steeper and faster into the true positive area, making less mistakes in all instances.

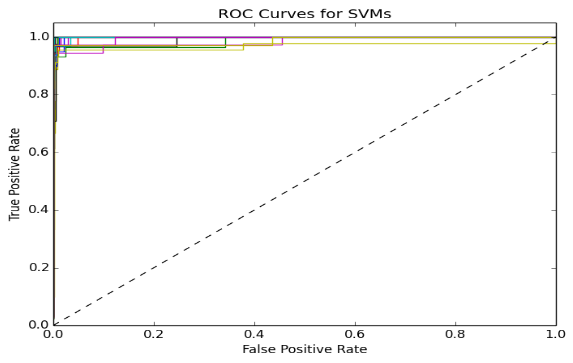

Figure 4. ROC curves of an SVC with linear kernel

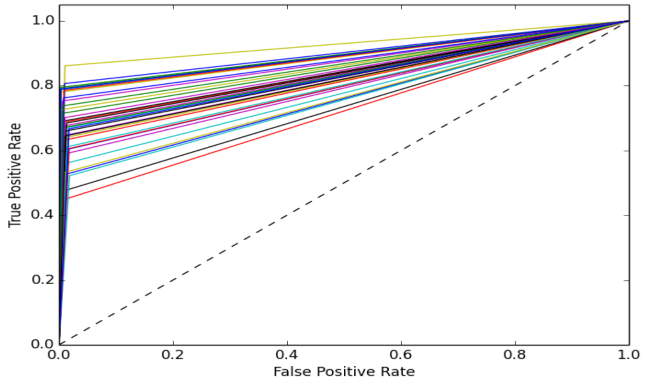

Figure 5. ROC curves of a Decision Tree with max depth = 100

Next we will be looking at the confusion matrices of SVCs with linear kernel and Decision Trees. We will appreciate this index in the way that it visually and automatically compares the accuracy of a method to another, and which are the characters that are most commonly mis-classified.

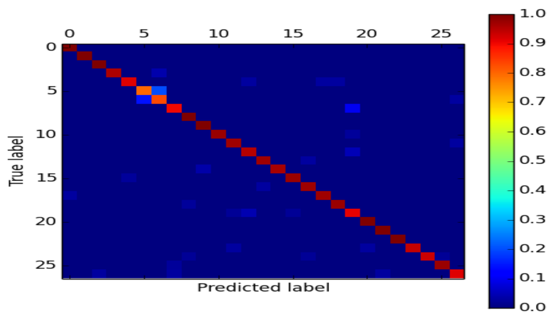

Figure 6. Confusion Matrix of an SVC with linear kernel

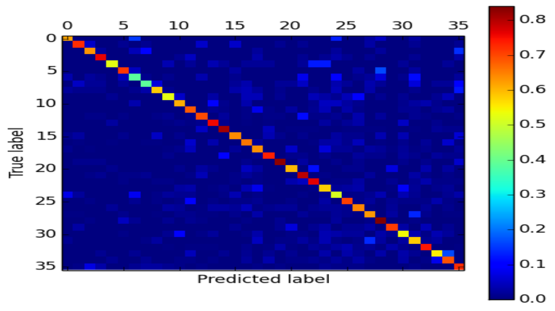

Figure 7. Conf. Matrix of a Decision Tree with max depth = 100

We can observe in these confusion matrices how the diagonal is much stronger than the sides, meaning that most of the instances of a certain class are classified correctly, and that there are very little false positives and false negatives. And it is visually clear how different the performance for these two methods is, because we can see a lot more light blue spots in the false positive area in decision trees for things that were mis-classified.

4. Results

4.1. Accuracy Conclusions

4.2. Real Time Demonstration

Character recognition was achieved with a very high accuracy and implemented in real time in a real environment. The methods and equations listed above have proven successful and reliable.
Here you can see a video of our interface, readings, character tracking, and an example of a ”Hello World” recognition: `https://www.youtube.com/watch?v=L0AaqIswKBE&list=UUEdPimfOpE81_NbsL7sXQhA&spfreload=10`

4.3. Next steps to product development

While very satisfied with the results, this is far from being a commercial product. But being that we could get this working in such a short period of time, I believe that with a couple of months of exclusive work on this devise, we could get into market a penn that implements Hidden Markov Models (While on our research we found that this algorithm could lead to smaller percentage errors) and has a very low percentage error, while at the same time is capable of doing some more advanced things like spaces, or math.

As for next steps to enhance the performance we could implement:

- Enhancing feature extraction by adding wavelet transforms, fft. Custom kernel that utilizes dynamic time warping.
- Additional learning on words for spell-checking. (eg. if you write ”word” and the pen classifies ”w” ”Q” ”r” ”d”) we can reclassify ”Q” to ”o”.
- More test subjects/bigger dataset.

Acknowledgments

Attitude estimation Kalman Filter and initial approximation provided by **open-source** code from Sebastian Madgwick (July 2012), based on his paper **Estimation of IMU and MARG orientation using a gradient descent algorithm**.

For trajectory reconstruction, consulted Jeen-Shing Wang, Yu-Liang Hsu, and Jiun-Nan Liu’s paper entitled **An Inertial-Measurement-Unit-Based Pen With a Trajectory Reconstruction Algorithm and Its Applications**.

Learning From Data by Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.T. Lin.