Authenticating Smartphone Users Via Accelerometer Readings

Andrew Giel
BS Stanford 2015, Computer Science
agiel@stanford.edu

Jonathan NeCamp
BS Stanford 2015, Computer Science
jnecamp@stanford.edu

***Abstract*—In the past two decades, using a person's gait as a biometric has been explored using the technologies of machine vision, floor-sensors, and personal accelerometers. Motivated by a Kaggle competition, this paper explores the use of the latter in an attempt to authenticate smartphone users via their gait biometric. In a smartphone abundant world, gait recognition as an authentication method has potential application as an alternative biometric to the more populized fingerprint sensor and eye-recognition. In this paper, the method of average cycle extraction through Dynamic Time Warping is used to create a template of a user's gait. Authentication is then dictated by the similarity (as determined by Dynamic Time Warping and least squares linear regression) of this template with the average cycle from a set of readings formed as the authentication request. This method is specifically powerful because it utilizes the periodic nature of a person's walking pattern. With this methodology and accelerometer readings provided by Kaggle, a correct authentication response was given around 60% of time. Although this is a significant improvement over simple guessing, the prevalence of errant authentications shows that this technique would need to be refined before any consideration is given to use as a smartphone security measure.**

I. Introduction

Smartphones with the capability for highly sensitive accelerometer readings have already proliferated the consumer market and are in widepsread use. Additionally, a study performed at MIT described ones gait as "an idiosyncratic feature of a person that is determined by, among other things, an individuals weight, limb length, footwear, and posture combined with characteristic motion" and concluded that a "gait can be used as a biometric measure to recognize known persons and classify unknown subjects"[1]. The combination of easy access to precise data and the distinctive nature of gait cycles presents the opportunity to identify and authenticate via acclerometer readings. Authentication via gait analysis allows for a non-intrusive biometric identification method with potential benefits in the forms of increased ease-of-use and security. We can model this sort of authentication as a paradigmatic problem in data science: binary classification. Such is the nature of this paper.

In this paper, we explore the possibility of authentication of smartphone users via accelerometer readings through a series of experiments and analysis. First, we review prior work in this field, and outline the methods used in the past. After explaining how we obtained our data, our implementation is outlined-beginning with data preprocessing techniques of resultant acceleration, time interpolation, and noise reduction. Cycle detection and average cycle designation, powered by Dynamic Time Warping, is examined in Average Cycle Extraction. Our authentication method wraps up the implementation portion. After establishing our method for performance evaluation, our experiments and results are shared and analyzed. Finally, we end with our closing remarks and hopes for future experimentation.

II. Review of Prior Work

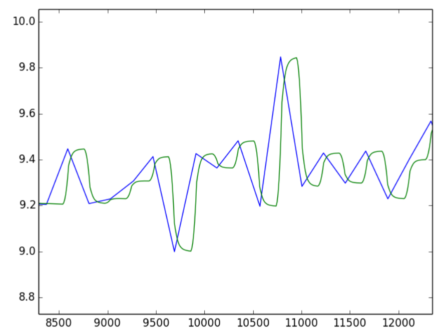
Biometric gait recognition is a topic that has been explored quite extensively over the last 20 years. Initial attempts at gait recognition were made using machine vision [2]. Additionally, the employment of floor sensors to record ones gait was also one of the earlier explored methods [3]. Within the last 8 years, wearable sensors in the form of accelerometers have been where a majority of the focus is for using ones gait as a biometric and is obviously most relevant to what we plan to explore. These various studies that specifically investigate using accelerometer readings present numerous methods for human gait detection. One of the more frequently come across methods consists of cross-correlating a set of test cycles with a template [4]. This involves identifying the individual steps from the gait cycle pattern, normalizing and averaging the steps and then computing the cross-correlation of this average with a template [4]. Hidden Markov Models have also been used for gait recognition and prove to be particularly effective at overcoming the irregularities of the recorded cycles [5]. Histogram similarity and cycle length are two more methods that produced minor error in gait detection [6]. A lot of studies also focused on the multitude of ways raw data can be preprocessed before use in authentication. Time interpolation, via linear interpolation, has been used to create fixed time intervals from irregularly timed readings [7]. Filtering, via weighted moving average filtering, can also be applied to remove the inherent noisiness of the recordings [7]. Although, many approaches have been taken to solving this problem, the sheer number of possible approaches allows for novel methods to be taken.

III. Data Collection

The accelerometer data was provided by Kaggle.com through the sponsor Seal Mobile ID. Seal collected the accelerometer readings through an app, published on Google's Playstore. An important caveat is that accelerometer readings were sampled in the background of a user's device. This adds an extra complication to cycle extraction as movements besides normal walking will be included in the samples given. Each data sample was composed of an acceleration measured in g's on the x coordinate, y coordinate, z coordinate, and the corresponding Unix time the reading was taken. Approximately 60 million samples from 387 devices were collected. Kaggle posted this data in pre-split files with one file marked for training and the other for testing.

IV. PREPROCESSING

Fig. 1. Raw (Blue) and Processed (Green) Data



Resultant Acceleration

Accelerometer readings initially consist of tuples that include the acceleration in the x, y, and z dimensions for an instance in time. Because the readings come from smartphones that can be oriented (in a persons pocket, hand, purse, etc.) in an infinite number of orientations, these coordinate labels are meaningless, Therefore, the readings were refined to just one, more meaningful resultant acceleration using the Euclidean norm:

$$r_t = \sqrt{x_t^2 + y_t^2 + z_t^2}$$

After this stage of preprocessing, data consists of resultant acceleration and time pairs.

Time Interpolation

Accelerometers are only actively recording data when they are experiencing a change in acceleration. Due to this protocol, the data is often inconsistently collected, with gaps in data reaching up to ten seconds. These large gaps and irregular time intervals in the data make detection of accurate cycles actually representative of a smartphone user's gait difficult. To account for this, sequences of data with gaps larger than a determined parameter (discussed in Average Cycle Extraction) were split into separate sequences and handled independently up to averaging all the cycles into one template. Inconsistent time intervals were then regularized using interpolation. Based off of experimentation and other studied attempts [7], it was determined that a regular interval of 10 milliseconds works well at providing enough granularity but not too many artificial data points. Linear interpolation was used, as it is the quickest method of interpolation and our granularity is small enough such that it approximates the values for small windows of missing data very well. This method is given, more formally by the equation

$$a = a_0 + (t_1 - t_0)\frac{(a_1 - a_0)}{(t_1 - t_0)}$$

where a, t is the new value, time pair and a_0, t_0 and a_1, t_1 are the previous and next data points respectively.

Noise Reduction

Accelerometer readings are inherently noisy. In order to remove this noise we decided to perform filtering as part of the preprocessing. For simplicity, we decided to implement and test the Moving Average and Weighted Moving Average methods with a window of 5. Moving Average is an averaging approach that reassigns a value based on its average with its surrounding points where the number of included surrounding points is dictated by the window size.

$$a_t = \frac{a_{t-2} + a_{t-1} + a_t + a_{t+1} + a_{t+2}}{5}$$

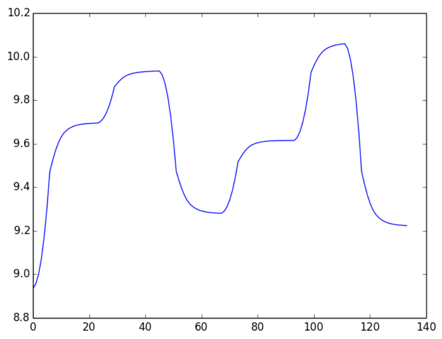
Weighted Moving Average works in a similar manner but assigns higher weights to surrounding points that are closer to the given point. Effectively, this allows for closer neigbors to have a larger influence in the average for a reasigned point than more distant surrounding points.

$$a_t = \frac{(a_{t-2} * 1) + (a_{t-1} * 2) + (a_t * 3) + (a_{t+1} * 2) + (a_{t+2} * 1)}{9}$$

In our final experiment, we chose to use Weighted Moving Average over the unweighted version, as we felt it gave a more accurate representation of the data while still smoothing and reducing the effect of outliers.

V. AVERAGE CYCLE EXTRACTION

Fig. 2. An Example Cycle Template



A person's gait is intrinsically periodic but it is not perfectly regular. To account for these two facets, accelerometer readings are processed by extracting each periodic cycle. These individual cycles can then be used to develop an average cycle that is ultimately the representative template of one's gait. Average cycle extraction is composed of multiple steps. First, the values of average cycle length and starting point for cycle detection are calculated. Then, using these values, all cycles are detected. Lastly, each detected cycles is combined to create an average cycle.

Determining Average Cycle Length and Start Point

Before cycles are detected, the helpful values of average cycle length, λ, and the starting point, μ, must be determined. The average cycle length is determined by working from the middle point, called the base point, of a sequence data. This is done, because the middle of a person's walk is more representative of their actual gait while end points (starting and stopping) represent extremes. From the middle, a baseline sequence of length 700ms is extracted. Moving forward one interval at a time, this baseline sequence is repetively overlaid up to 1900ms past the baseline start point. From research [8], 1900ms was found to be the maximum and 700 the minimum lenghth of a person's cycle. The difference in time between the point that yields the minimum distance as determined by manhattan distance and the base point is λ_1. This is repeated going backward to determine λ_2. λ is then simply the average:

$$\lambda = \frac{\lambda_1 + \lambda_2}{2}$$

Next, the starting point for cycle detection is determined. A consistent starting point for all sequences of readings is important for maximum overlap when determining the average cycle. It was chosen to always start from a local minimum. Once again, with extremes at the beginning and end, cycle detection will begin in the middle. Moving to the middle of a sequence, the nearest local maximum is found. From here the desired local minimum can either be to the right or left of this max. Because, gaits are composed of a left step and right step that can possibly differ from each other, it is desired to start from the local minimum of the same step for all sequences. Therefore, four different checks are used to increase the likelihood of choosing the minimum of the same step for all sequences. The checks determine whether the minimum to the right or the left of the maximum is chosen as the starting point. These values of λ and μ are next used to detect cycles.

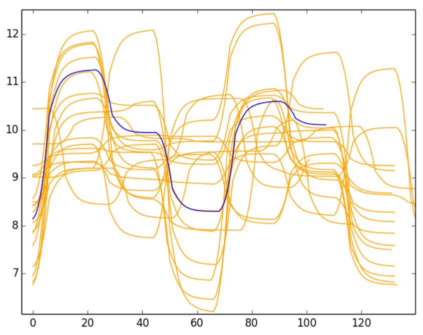
Cycle Detection

Cycle detection is a recursive process. Starting at μ, the end point is found by moving one cycle length and then searching for the local minimum. This point becomes the new start point and cycles are recursively found by the same procedure of moving λ and finding a minimum. This is done in the forward direction until the end of the data sequence has been reached and then in the backwards direction. Once the cycles are ascertained, they are used in the next step to determine an average cycle.

Average Cycles

In order to create an average cycle from the set of extracted cycles, we must take into account that these cycles, even when from the same device, vary significantly in terms of length. This means simply taking the mean of these cycles can be ineffective and inaccurate. To address this challenge, we use the Dynamic Time Warping Algorithm to get an intuition for how similar two cycles are and then designate the cycle with the smallest average difference over all other cycles as our template cycle. Unfortunately, since Dynamic Time Warping is an $O(N^2)$ algorithm, and comparing each cycle to all other cycles is also $O(N^2)$, this process can be quite expensive. In order to approximate this template extraction, we sample K=20 cycles from the set of all cycles. This random sampling allows for selection to be quick and relatively inexpensive, but still utilize the power and flexibility of the Dynamic Time Warping (DTW) Algorithm.

Fig. 3. An Average Cycle (Blue) Extracted from Sample Cycles



Dynamic Time Warping

The Dynamic Time Warping Algorithm serves as a way to quantify the difference between time dependent sequences, where the two sequences can differ in both their length and regularity of readings. In our implementation, DTW is not used until after the pre-processing has occurred, meaning the time interval between data points will be standardized by interpolation yet the length of the sequences DTW sees can vary dramatically.

DTW operates by creating a minimum cost matrix $C \in \mathbf{R}^{m \times n}$ representing the difference between two sequences a and b where $m = |a|$ and $n = |b|$. This is accomplished similarly by replicating the behavior of the Levenshtein Edit Distance algorithm. Each point i, j where $1 \leq i \leq m$ and $1 \leq j \leq n$ within C represents the cost of transforming a i length sequence of a to a j length sequence of b, or visa-versa. The cost is defined by some function D and the minimum cost of a Levenshtein edit (either an insertion, deletion, or substitution) of prior indices within the sequences. We defined D as simply the absolute value of the difference between the two sequences, $D(i, j) = abs(a_i - b_j)$. The edit distances are defined based on the values of C for prior indices: subsitution is $C_{i-1,j-1}$, insertion is $C_{i,j-1}$, and deletion is $C_{i-1,j}$. Subsequently,

$$C_{i,j} = D(i, j) + min(C_{i-1,j-1}, C_{i,j-1}, C_{i-1,j})$$

The result of running DTW as such is a fully populated C. Within C one can define the DTW score to be $C_{m,n}$. Additionally, one can define a path $Path_C$ of length k from $C_{0,0}$ to $C_{m,n}$ representing the path of least "resistance" when transforming a to b. This can be visualized as a path navigating from one corner of the matrix to the opposite corner, never moving backwards in time. From this path, we can create what will be referred to as a difference-development line Dev_C, a monotonically increasing line where

$$Dev_{Ci'} = \sum_{i=0}^{i'} Path_{Ci}$$

where $0 \leq i' \leq k$. As such, DTW gives us both a score representing the difference btween two cycles, and a line representing the sum of how different they are over the length of the two cycles.

Algorithm 1: DTW
Input ← Time-Sequence A, length M
Input ← Time-Sequence B, length N
Output → DTW Score
Output → $Path_C$ Path of least resistance through C

```
DTW = array(n, m)
for i in [1, m]:
    DTW[i, 0] = -inf
for j in [1, n]:
    DTW[0, j] = -inf
DTW[0, 0] = 0
path = []
for i in [1, m]:
    pathOptions = []
    for j in [1, n]:
        cost = D(a[i], b[j])
        edits = [DTW[i-1][j-1],
                DTW[i][j-1],
                DTW[i-1][j]]
        paths = [(i-1, j-1), (i, j-1), (i-1, j)]
        minEditIndex = edits.indexOf(min(edits))
        DTW[i, j] = cost + edits[minEditIndex]
        pathOption = (edits[minEditIndex],
                        paths[minEditIndex])
        pathOptions.append(pathOption)
    pathCost = min(pathOptions, lambda x: x[0])[0]
    path.append(pathCost)

return DTW[m][n], path
```

The template ascertained by taking the lowest average DTW score is now considered the template cycle, denoted T_{ID} for device ID and will ultimately be used in the authentication stage to ascertain if the given test cycles are similar enough to that of the T_{ID}. Also necessary for authentication if Dev_{avg}^{ID}, the average Dev_C for all cycles against the T_{ID}.

VI. Authentication

The task presented to us by the Kaggle Competition is one of identification via accelerometer biometrics. In order to evaluate our method, the test data is presented in a specific manner to mirror that of an actual authentication process. Presented with a sequence of accelerometer data and an ID from our training data, our task is to determine whether or not this new test sequence is from the same person/device as the ID supplied. One can think of this as a binary classification- either this new sequence presented is from the supplied ID (in which case we authenticate/give a positive assessment), or this new sequence is not from the supplied ID (in which case we deny/give a negative assessment).

Our strategy for this binary classification involves using our cycle template to determine how similar the gait of the test sequence is to that of the train sequence for the supplied ID. The first step is to extract an average cycle A from the authentication sequence, using the same methodology as was used during the creation of the device template cycle T_{ID}. Next, DTW is run between T_{ID} and A, returning a score representative of the difference between the two and a difference-development line Dev_A.

Now the difference-development lines for both the training set (Dev_{avg}^{ID}) and the test set (Dev_A) are available for analysis. Running linear least-squares regression against Dev_{avg}^{ID} results in a line in the form of $y_{train} = m_{train}x + b_{train}$ where m_{train} is the slope and b_{train} is the y-intercept.

This process is repreated with the test data. Running linear least-squares regression against Dev_A results in a line $y_{test} = m_{test}x + b_{test}$.
At this point, linear regression has created two lines, one representing the difference development in the training data and the cycle template and one representing the difference development of the test data and the cycle template. Only one feature of these lines is used to determine if the test dataset is from the same person as the train dataset, the slope. Comparing m_{train} and m_{test}, if m_{test} is within a threshold percentage difference of m_{train} or $m_{test} < m_{train}$, then authentication occurs. If $m_{test} > m_{train}$ by a threshold percentage, then access is denied, or that request in not authenticated. Authentication occurs if the test data cyles diverge from the cycle template T by a theshold percent τ greater than the divergence of the train data or less than that of the train data, otherwise we deny access.

We experimentally determined our τ = 3%, as outlined in the Experiment and Results section. We found that this is the threshold at which our algorithm performs best, as lower thresholds have greater False-Negative rates, and higher have greater False-Positive rates.

As a quick summary, in order to authenticate a sequence of test data the slope of difference-development between the test cycles and the cycle template and the slope of the difference-development between the test data and the cycle template are compared. If the difference in the slopes is less than a threshold percent difference or then authentication, otherwise access is denied.

VII. Evaluation

In order to evaluate the effectiveness of our implementation, we are going to evaluate using a Receiver Operating Characteristic curve, or ROC curve. This is a curve representative of our binary classifier's performance, as it plots true-positive rate against false-positive rate as thresholds change. Since we are not going to be changing thresholds (making one set of predictions per test set), our ROC curve will consist of 2 vectors $\in \mathbf{R}^3$, namely x = [0, FPR, 1] and y = [0, TPR, 1] where

$$FPR = \frac{numFalsePositives}{(numFalsePositives + numTrueNegatives)}$$

$$TPR = \frac{numTruePositives}{(numTruePositives + numFalseNegatives)}$$

The area underneath the ROC curve serves as our performance metric. This is the method by which Kaggle (the source of our data) evaluates submissions to the competition, and it serves well for our task of authentication since it weight true positives favorably (being able to authenticate when its actually you) and false positives very negatively (having others authenticate when its not you). In order to calculate the area under the ROC curve, we must evaluate the integral

$$A = \int_{-\infty}^{\infty} y(T)x'(T)dT = \int_{-\infty}^{\infty} TPR(T)FPR(T)dT$$

where $TPR(T)$ and $FPR'(T)$ are the ΔTPR and ΔFPR between indices of T in x and y. The resultant A, where $0 \leq A \leq 1$, will be the metric by which we evaluate performance.

VIII. Experiments and Results

Our first experiment involved splitting the Kaggle-supplied training data for each device, using one half to train on and the other half to test against. This would be considered a training run, where we could evaluate our performance with smaller data and the ground-truth answers. For each device, we created 10 test authentications: 1 that matched the device that should be authenticated and 9 that did not match the device and should be denied access. These 9 devices were randomly sampled from the set of all other devices' test sequences. We ran our algorithm, training on the half-sized training sets per device and testing against the 10 authentication questions. In order to determine the threshold τ which we wanted to use on the full Kaggle run, we ran this experiment mulitple times, with tau in the range of 0% to 50%. Outlined below are the results of 1%, 3%, and 5%.

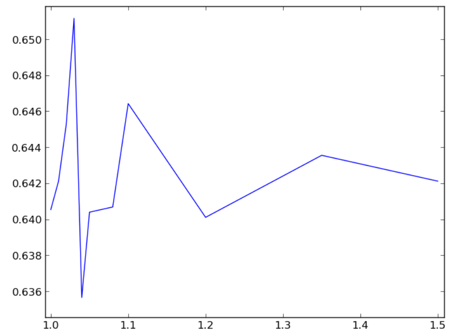
TABLE I. Test Run - Half Size Train Sets

	⊘	**T**	**F**
τ =1.01	**P**	265	1395
	N	2088	122
	⊘	**T**	**F**
τ =1.03	**P**	267	1350
	N	2133	120
	⊘	**T**	**F**
τ =1.05	**P**	267	1425
	N	2058	120

From these experimental results we were able to create an ROC curve for each τ, and found our highest area under ROC to be: 0.65 for τ = 3%. Thus, we determined our τ to be 3% for the full test run.

Our second experiment was to run against the Kaggle authentication set, or a "full" run. This involves training for each device on the entire training set supplied by Kaggle. Once each device had been trained and had a template T_{ID} established, the Kaggle questions were addressed. Each question, of which there are 90,024, consists of a question ID, device ID, and sequence ID. The "question" being posed is whether the sequence corresponding to the sequence ID should be authenticted against the corresponding device for the device ID. Each question represents an authentication request. After running our algorithm, and submitting to Kaggle, we received a result of .61365, the area under the ROC curve for our test run.

Fig. 4. Area Under ROC against 1.00 + τ



IX. Analysis

Our results show moderate success in our task. We were able to successfully authenticate and deny approximately 65% during training runs, and 61% during test runs. Unfortunately, our success on this dataset does not indicate that this method would be a viable means of authentication for smartphone users as stands. This is due to the nature of an authentication task. In order for a method to be used successfully for authentication, it must achieve a high True-Positive Rate and a very low False-Positive Rate. Although we found True-Positive Rates greater than 75% with our implementation, our False-Positive Rates were far too high: usually in the rangeof 35-40%. This means that although you would most likely be able to log in to your own phone simply by walking, so would 3 or 4 out of 10 random others, making our system convenient but not secure.

The major detractor from our algorithm's effectiveness is the inconsistency of data captured via smartphone accelerometers. Accelerometers, as mentioned previously, collect data whenever they experience a change in acceleration. For a sensitive accelerometer within a smartphone within an individual's pocket, this is very frequent and not often associated with the individual walking. Subsequently, a lot of extraneous data not associated with an individuals gait is collected. This forces the cycle detection algorithm to be very selective on which time sequences it parses and determines to be a gait cycle. For some equences even, not a single cycle was able to be detected and authentication was automatically denied. Even then, once a cyclic or periodic time sequence is identified, there is a lot of variance in terms of the amplitude and shape of these cycles. This is due once again to the ubiquity of smartphones. Users carry smartphones at nearly all times: when walking, jogging, running, biking, sober, or intoxicated. All these activities result in cyclic time sequences, yet result in drastically different accelerations. Subsequently, the process of determining the average cycle from a sample of these cycles can result in average templates that do not necessarily reflect the gait of an individual when walking normally, but instead perhaps when doing another task or when walking atypically. This results in average development-differences that can be relatively high, as cycles all from the same individual are behaving so drastically different. High average development-differences (Dev_{avg}) lead to a high threshold of difference for a sequence to be authenticated, leading to False-Positives. These cycle amplitude differences also result in difficulties in the authentication process of legitimate users. Since a user can display such varying types of gaits depending on situation, users attempting to authenticate their own device may have difficulties: as their current accelerometer data may not reflect that which the algorithm expects. This reflects a False-Negative authetication.

If this system were to be implemented in actual smartphones, we believe the best way to do so would be to have a designated training period. Instead of forcing the training portion of the algorithm to decipher the characteristics of an individual's gait from all accelerometer readings, designating a set of cycles as normal walking cycles would be preferable. This would decrease the variance in the training data- creating better-defined cycles (and subsequently more accurate templates) and leading to shallower difference-development lines. These difference-development lines would give less variance in terms of what would be authenticated, and would undoubtedly result in a lower false-positive rate. This designated training period would be achievable from a user-experience perspective by requiring the user to walk a certain distance when setting up the accelerometer based authentication on their device. We feel that the lack of such a procedure for the data this paper used harmed the performance of our imlpementation in these experiments.

X. Conclusion

We implemented a technique for authenticating smart phone users via accelerometer readings based on gait cycle detection and templating. Our conclusion after testing the algorithm's effectiveness on data supplied by Kaggle, is that this technique although generally effective for identification, is not secure enough to be used for authentication. We do believe that this method, or appraoches similar to it, may be used in the future for identification or authentication tasks, as gait recognition shows promise to be a convenient and unique biometric form of identity.

Acknowledgment

The authors would like to thank Arun for his guidance. Also, we would like to thank Coupa Cafe for their delicious coffee.

References

[1] Lee, Lily. "Gait analysis for classification." (2003).

[2] Niyogi, S.A.; Adelson, E.H., "Analyzing and recognizing walking figures in XYT," Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on , vol., no., pp.469,474, 21-23 Jun 1994 doi: 10.1109/CVPR.1994.323868

[3] Orr, R. J. Abowd, G. D. 2000. The smart oor: A mechanism for natural user identication and tracking. In CHI 00: CHI 00 extended abstracts on Human factors in computing systems, 275276, New York, NY, USA. ACM Press.

[4] Heikki J. Ailisto ; Mikko Lindholm ; Jani Mantyjarvi ; Elena Vildjiounaite ; Satu-Marja Makela; Identifying people from gait pattern with accelerometers. Proc. SPIE 5779, Biometric Technology for Human Identification II, 7 (April 05, 2005); doi:10.1117/12.603331.

[5] Nickel, C.; Busch, C.; Rangarajan, S.; Mobius, M., "Using Hidden Markov Models for accelerometer-based biometric gait recognition," Signal Processing and its Applications (CSPA), 2011 IEEE 7th International Colloquium on , vol., no., pp.58,63, 4-6 March 2011 doi: 10.1109/CSPA.2011.5759842

[6] Gafurov, Davrondzhon, Kirsi Helkala, Torkjel Sndrol. "Biometric Gait Authentication Using Accelerometer Sensor." Journal of Computers [Online], 1.7 (2006): 51-59. Web. 1 Oct. 2013

[7] Derawi, M.O.; Nickel, C.; Bours, P.; Busch, C., "Unobtrusive User-Authentication on Mobile Phones Using Biometric Gait Recognition," Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 2010 Sixth International Conference on , vol., no., pp.306,311, 15-17 Oct. 2010 doi: 10.1109/IIHMSP.2010.83

[8] Holien, Kjetil. "Gait recognition under non-standard circumstances." Master's thesis, Gjvik University College-Department of Computer Science and Media Technology (2008).