IMPROVING GESTURE RECOGNITION PERFORMANCE USING THE DYNAMIC SPACE-TIME WARP ALGORITHM

by

DANNY ALLEN HANSON

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2013

To my wife and son who are the constant guiding light in my life.

ACKNOWLEDGEMENTS

As a working adult, the work presented within represents several years worth of effort to complete. During this time, many people supported and inspired me. I wish to thank Dr. Vassilis Athitsos for his continued support though the years. Without the support of my family, Shelley and Patrick, this work would have never been completed. Additionally, many of my co-workers supported my efforts including: Brendan Drew, Dr. Jason Luck, Charles Carpenter, Chris Baker, and Ed Moon. These individuals listened to my ideas and allowed me to find my own path. Finally, I would like to thank my employer for covering the costs and allowing me the time and resources to finish this work.

April 12, 2013

ABSTRACT

IMPROVING GESTURE RECOGNITION PERFORMANCE
USING THE DYNAMIC SPACE-TIME WARP ALGORITHM

Danny Allen Hanson, M.S.

The University of Texas at Arlington, 2013

Supervising Professor: Vassilis Athitsos

The DSTW algorithm was originally used as the fundamental algorithm for a gesture recognition software. When the need arose for implementing gesture recognition on-board a robotic vehicle, the original recognition software needed to undergo several changes in order to meet the requirements of the target platform. The original software was written in Matlab and had to be ported into a native language in order to operate on the new platform. To support experiments needed to select a distance and τ function, the new code needed to be designed to support dynamic binding of distance and transition (τ) functions. The software needed to handle over 140 experiments to determine the appropriate distance and τ functions. A new classifier based on the A^* algorithm was proposed and implemented to further reduce runtime performance, and a new τ function based on template matching between the various candidates provided by the detector was proposed and implemented. This work covers the results of theses efforts in Improving Gesture Recognition Performance using the Dynamic Space-Time Warp Algorithm.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

Introduction

In the domain of gesture recognition, there are several methods commonly used to recognize and classify gestures [2]. In some methods, the position of the hard must be tracked between frames in order to correctly classify the gesture. This approach requires a strong tracker in order to obtain acceptable recognition performance. An alternative approach which attempts to remove the requirement of a strong tracker was presented in Alon et. el. [3]. The approach specifies a way of utilizing a weak detector to feed a dynamic programing table which would compute the lowest cost alignment between a known exemplar and a sample query. The core idea being that a weak detector may not label the gesturing hand as the highest scored detection, but the hand would usually be detected within the first k candidates. If an algorithm could accept all of the presented candidates and determine the lowest cost warping path between and exemplar and a sample signal, then the weak detector would provide useful results. The Dynamic Space-Time Warping algorithm was designed to allow multiple candidate detection for each frame within a video sequence. The Dynamic Space-Time Warping algorithm was later improved by Alon et al. [4].

The algorithm provides a method of computing the warping path which provided the lowest cost alignment between an exemplar signal and a sample signal which is composed of various candidates at each time within a sequence. By design, the algorithm returns a mapping which provides the best possible alignment of the specific candidate at each point in time to the exemplar signal. The method was shown to produce good results in the domain of gesture recognition when using a small alphabet composed of the Palm Graffiti 0-9 digit set, shown in Figure 5.1 [3].

The DSTW algorithm utilizes two important functions to produce the best warping path: a distance function and a transition (τ) function. The distance function computes the distance between an exemplar feature and a sample feature. The transition function (τ) is used to compute the cost of transitioning between the states within a dynamic programing matrix. The result of the distance function is combined with the result of the τ function to determine the lowest cost warping path between the exemplar and the sample.

1.1 Motivation

The following work was inspired by searching for improvements to the results provided by the original DSTW algorithm. Specifically in the areas of improved gesture recognition and reduced computation performance. A reduction in runtime was required to support the use of the DSTW gesture recognition system on-board a robotic vehicle. The original DSTW results were obtained using software written in Matlab and did not provide sufficiently fast runtime performance to be used on-board an active robot.

After a significant reduction in runtime was achieved, attention was re-focused on improving the recognition of the system. For the intended application, the recognition system would now have to handle issues stemming from having the camera mounted on a non-fixed platform. Some of these issues include scale and translation variance of the gesturing subject and recognition within a cluttered scene. With the gains in runtime, it was now possible to use additional computation to improve positive classification of gestures and still maintain runtime requirements.

1.2 Contribution and Scope

Within this work, the main focuses are divided between increasing runtime performance (in terms of reduced wall time measurements) and increasing correct recognition rates for the gesture recognition components. Runtime performance was addressed by redesigning the core software. In addition to the redesign, a new classifier based on the A^* algorithm was developed to further reduce system runtime. In order to improve recognition rates, a new τ function was designed and tested.

In order to reduce the runtime and conduct the necessary experiments within the time constants, the original gesture recognition system had to be completely redesigned and reimplemented using a native language that utilized computationally efficient data structures and programming techniques which included multi-threading and smart memory management. The resulting design allowed the required flexibility to perform the needed experiments and the reduced runtime to complete all of the 148 experiments in a timely manner.

This work also introduces a new classifier which was based on the A^* algorithm. The classifier was able to reduce the search time needed to classify incoming gestures. The proposed classifier works with each class traversing the exemplars until the algorithm reaches the end node of the sample's class. The class for which the algorithm reaches the end node first is reported to be the class of the presented sample.

Based on the analysis of the native softwares' reduced runtime performance, this work presents a method of utilizing the multiple candidates presented by the weak detector to the DSTW algorithm. The method utilizes template matching in an attempt to influence the dynamic programing matrix used by the DSTW algorithm. By matching a template image chip within the τ function, the DSTW algorithm should naturally find more consistent warping paths and thereby lead to an improvement in recognition performance.

1.3 Organization

This work is organized into distinct sections covering the phases of development. In Section 2, background information is presented to introduce the reader to the fundamentals upon which this work builds. In Section 3 titled ”Proposed Optimizations,” several methods of enhancing the recognition performance and reducing the runtime performance of the original DSTW based gesture recognition software are presented. Section 4, ”Implementation”, details the software development of the replacement gesture recognition system. The results of the experiments performed are discussed and analyzed in Section 5. Proposals for future work can be found in Section 6.

CHAPTER 2

Background Review and Related Work

This work attempts to increase the performance of a gesture recognition system to the point were the software can be used on an active robotic platform for the recognition of control gestures. Primarily, this work is an implementation driven continuation of the work done by Alon et al. [3]. To that extent, the same datasets were used and the recognition alphabet of the Palm Graffiti digits from 0-9 was utilized. All experiments designed for use within the context are similarly in nature to those developed by Alon et al. [3]. In this section, the reader is presented with some background regarding the fundamental techniques and background.

2.1 Related Work

A significant body of work has already been done within the domain of gesture recognition. This work largely followed the related work of Alon et al. in "Simultaneous Localization and Recognition of Dynamic Hand Gestures" [3]. Of direct importance is Chen et al. [5] work in real-time gesture tracking and hidden Markon models. Black and Jepson's work provides a foundation for gesture recognition [6]. Yuan et al. provides an outline of features used to detect hand positions within a single image frame[7].

2.2 Dynamic Space-Time Warp Algorithm

The Dynamic Time Warp (DTW) algorithm uses a dynamic programing technique to compute the similarity between two sequences of features which are represented as vectors such that each index is a specific point in time [8]. The algorithm was originally applied in the domain of speech recognition [9, 10] and is now commonly used in the context of gesture recognition [11, 12]. The algorithm compensates for variability of the sample rate of the signals under comparison and aligns the various points in such a manner to reduce the distance cost of the alignment. The return value of the algorithm is the best possible similarity between the given signals. A trivial extension to the core algorithm also returns the warping path of the alignment.

The Dynamic Space-Time Warping (DSTW) algorithm is based on the DST algorithm with the extension that one of the signals, known as the query, may have multiple candidates at each index within the sequence. The DSTW algorithm leverages the multiple candidates to determine the best possible alignment to the reference signal by choosing the closest candidate for each point in time. The DSTW algorithm removes the requirement of perfect detection as needed by the DTW algorithm. In removing this requirement, the DSTW algorithm is able to determine the most likely candidates which compose the signal, thus allowing the detector to present a list of possible detections which reduces the detector's accuracy requirements.

The DSTW algorithm is influenced by two important functions: Distance (d) and Transition (τ). The distance function determines the similarity between a feature from the sample signal and a feature from the exemplar signal. The τ function is used to determine the cost of transition between two states within the dynamic programing matrix. The definition of these functions defines the performance characteristics of the DSTW algorithm.

2.2.1 Features for Gesture Recognition

In the domain of gesture recognition, the feature vector presented in Alon et al. [3] was defined as follows:

For every frame j of the query sequence, K candidate hand regions are found as described in the previous section. For every candidate k in frame j a 4D feature vector $Q_{jk} = (x_{jk}, y_{jk}, u_{jk}, v_{jk})$ is extracted. The 2D position (x, y) is the region centroid, and the 2D velocity (u, v) is the optical flow averaged over that region.

The challenge in using the original features are introduced in the two-dimensional component, (x, y). When calculating the centroid of the moving object, erroneous noise surrounding the detection must be handled appropriately. If noise is grouped together with the detection, then the centroid is artificially increased. If extended to the logical extreme of equally distributed noise over the entire image area, the centroid reduces to the center of the image which will not provide an adequate feature for correctly matching the sample signal to an exemplar.

2.2.2 Manhattan Distance Function

A common distance function which may be used when measuring the distance (or similarity) between two vectors is the Manhattan distance function. In the original paper, the authors chose the Manhattan distance function for use with the DSTW algorithm. The function computes the distance between two n-dimensional vectors as shown in Equation 2.1.

$$d(\bar{x}, \bar{y}) = \sum_{k=1}^{n} |x_k - y_k| \tag{2.1}$$

Use of the Manhattan Distance function presented a few issues when used with the DSTW algorithm in the domain of gesture recognition. One of the issues that was discovered during initial testing was false measurements when the capture resolution of the exemplars and samples differed greatly. Given that the vectors' units are image pixels, then if the sample image and exemplar image are captured using different resolutions, the Manhattan Distance function will falsely penalize a matching signal. Another problem discovered during test was related to the whole units that measured the distance. The Manhattan distance function would measure a two-pixel diagonal offset with a value of 4, when Euclidean geometry tells that the value should be 2.83. Occasionally, in testing the error would build up to the point where it was significant enough to result in a mis-classification.

2.2.3 Manhattan τ Function

In Alon et al., the originally presented τ function is shown in Equation 2.2 [3]. The τ, or transition function, is used to determine the cost of transitioning between two states within the dynamic programing matrix. In the DSTW algorithm, the function determines which candidates should correspond to which features of the exemplar in order to produce the lowest cost alignment between the sample and the exemplar. The presented τ function did not to utilize any information with regard to the candidate when considering the cost of transitioning.

$$\tau(\omega, \omega') = \begin{cases} 0 & \text{if }_i \text{ or }_j = 0 \\ |\omega_i - \omega'_i| + |\omega_j - \omega'_j| & \text{otherwise} \end{cases} \tag{2.2}$$

During testing of the original gesture recognition software, this limitation of the τ function would allow a candidate consisting of noise to align more closely spatially than an adjacent candidate containing a hand that was slightly further away. Enabling a lower cost alignment using spatially closer noisy candidates leads the DSTW algorithm to produce an artificially lower distance cost between an exemplar and the query. The lower distance cost was determined to be a contributing factor to mis-classification leading to a lower effective recognition rate.

The DSTW τ function accepts a pair of mapping vectors, ω which defines the current state within the dynamic programing matrix, and ω‘ which is the next candidate state under consideration for transition. Using the mapping vectors, the τ function effectively knows which features within the exemplar and the sample are involved in the transition.

2.3 Search Algorithms

A fundamental goal in the domain of gesture recognition is to determine the classification of the sample signal. One method of utilizing the DSTW algorithm for use within a classifier is to construct a database of known exemplars which can be searched for a matching exemplar. The classification of each exemplar stored within the database is known *a priori*. The process of determining the classification of the sample signal begins with computing the DSTW similarity between each sample and suspected exemplar stored within the database. The pairing of a similarity operator with a search method builds a functional classifier. In the original paper, Nearest Neighbors was chosen as the search algorithm. A another search algorithm that is popular in the path planning domain is A^* [13].

2.3.1 Nearest Neighbor Classifier

A nearest neighbor classifier is an example of one such search method. The algorithm is a simplification of the *(k, l)* Nearest Neighbor Classifier [14, p. 500-501] where *(k, l)* is defined as*(1, 0)*. In conjunction with the DSTW algorithm, the similarity score for each exemplar is determined. The sample is reported to have the same classification as the exemplar with the lowest score. In this case, DSTW is executed between the sample and every exemplar member of the database. However, only the exemplar with the lowest score must be retained during the execution.

A nearest neighbor classifier provides a good degree of accuracy at the expense of additional computational requirements. The classifier can produce poor results in the event that a poor exemplar or a mis-classified exemplar is added to the database. This introduces a requirement to ensure that only the best exemplars are added to the database. In many applications, the quality requirement is a significant barrier to entry.

2.3.2 A^* Algorithm

A^* is a common minimizing search algorithm often used to reduce search time for large datasets [15, p. 97-101]. A^* is implemented as a directed search using the cost equation shown in Equation 2.3 where $f(n)$ represents the estimated cost between the node n to the goal node, $g(n)$ represents the cost between the n node and the start node, and $h(n)$ is a heuristic function which represents an estimated lowest cost path between the n node and the goal node.

$$f(n) = g(n) + h(n) \tag{2.3}$$

A^* has been used with a great degree of success in the domain of path planning. The performance of the A^* algorithm is determined by the heuristic function. The heuristic functions guides the search and allows the algorithm to focus on the most likely lowest cost path to the goal. An admissible heuristic is a function which never over estimates the costs of the path between the n node and the goal node. Given that the function $h(n)$ is admissible, the A^* algorithm is complete and optimal [15, p. 97]. The challenge when using an A^* algorithm results from having to define the heuristic $h(n)$ function in an admissible way.

The opposite to an admissible heuristic is an inadmissible heuristic. When an inadmissible function is used as a heuristic with the A^* algorithm, an optimal solution is no longer guaranteed by the algorithm. When using an inadmissible heuristic function, the final results may be a local minimum within the search space. However, an inadmissible function will usually execute with a reduced runtime and may provide results that are good enough for the problem domain.

CHAPTER 3

Proposed Optimizations

Within the following sections, proposed optimizations for to the DSTW gesture recognition software are presented. The section on Runtime Performance (3.1) attempts to address the limitations related to the original implementation of the DSTW gesture recognition software. Additionally, Improved Features for Gesture Recognition Section 3.2, proposes an enhanced set of features to be used with the DSTW algorithm's distance and τ functions.

Section 3.3 entitled Relative Distance Function explores a new coordinate system to address issues stemming from the use of image pixel coordinates in gesture recognition. Template Matching τ Function introduces a new method of utilizing information regarding each candidate in the query for the purpose of effecting the transition function. A new classifier is introduced the Section 3.5 entitled A^* Nearest Neighborhood Classifier which utilized the power of an A^* search algorithm to classify sample signals.

3.1 Runtime Performance

The original implementation of the DSTW gesture recognition software was written in Matlab. Matlab provides an easy to use, rapid development platform for the construction of computer vision algorithms. However, Matlab lacks the performance of a native programming language. In order to build a software system capable of performing numerous experiments within a limited about of time and to have a working software solution for use on a robotic platform, the original software needed to be rewritten using a native language. The language of choice for this work was C++, utilizing reusable components from Boost and OpenCV open-source libraries.

3.2 Improved Features for Gesture Recognition

Starting with the original feature vectors given in Alon et al. [3], a few enhancements were proposed. The first is the inclusion of a bounding box containing the face of the subject who is performing the gestures. The bounding box of the face allows the location of the hand position to now be referenced in a coordinate system with the origin located at the center of the face. The bounding box of the face provides information that can be used to introduce a degree of scale invariance with regard to computing the hand size of the subject performing the gesture.

In the original DSTW gesture recognition software, an assumed value for the size of the hand within the image was encoded into the detector algorithm. Using the face bounding box, the software may estimate the size of the hand as one-half the size of the face. As the subject's distance to the camera increases, the size of the face, and proportionally the size of the hand decreases. By addressing the scale invariance issue, the centroid defining the hand blob can be successfully reduced. The reduction of the area also reduces the amount of noise surrounding the hand that may influence the computation of the centroid location.

3.3 Relative Distance Function

With the feature set extended to include the face bounding box, the (x, y) position can now be defined as (x', y') whose values are defined by Equation 3.1. Improvements in translational invariance are made when (x', y') are stored as relative values to the center of the face. The improvements are a result of reducing the effect of an offset subject performing the gesture. The proposed change does not account for the individual subjects' arm length, and as such does not address all transitional invariance issues.

$$\begin{aligned}&\texttt{Let } b \texttt{ be a bounding box, then:}\\&(x', y') = (x, y) - (b_x + \tfrac{1}{2}b_{width}, b_y + \tfrac{1}{2}b_{height})\end{aligned} \tag{3.1}$$

3.4 Template Matching τ Function

Given the multiple candidate nature of the DSTW algorithm, there must exist a method of utilizing the additional information from each of the candidates to select the most consistent warp path. The detector used for the original DSTW gesture recognition produces candidates by selecting the top k scores from the result of Equation 3.2 [3]. In the presented equation, the $P(motion)$ component was calculated using image subtraction [16, p. 253-4]. $P(skin)$ was calculated using a 3D histogram trained for skin detection. The histogram used red, green, and blue pixel components for each dimension, containing 32 bins. Each bin contained the probability of skin for the given R, G, B combination. The histogram was trained manually as part of another effort.

$$P(hand) = P(skin) \times P(motion) \tag{3.2}$$

Since the top scores from Equation 3.2 may vary between frames, selecting the top k candidates many not yield a consistent result vector between frames. Therefore, it is not possible to assume that candidate k_1 in frame n is the same object as candidate k_1 in frame $n + 1$. The limitation of the original detector may have been the reason that the candidate information was not previously included as part of the τ function.

A proposed solution to allow the candidate information to be considered in the τ function is to use the stored template image of the hand region to match between various candidates. Transitions between candidates with a strong template matching score between their perspective hand regions at each time interval are more likely to represent the path through space which the hand traveled. Given a sufficient frame rate for the sampled video, when two candidates from adjacent time intervals present a strong match, the probability of the two objects representing the same image region within the video is very high. Whereas, when the template match is weak, the probability is significantly lower.

3.5 A^* Nearest Neighborhood Classifier

Based on the A^* path planning algorithm, the A^* Nearest Neighborhood Classifier uses a heuristic function to traverse a priority queue of defined nodes. The node is a data structure which contains a score value used for sorting and selection, the name of the class for which the node represents, and an iterator to the next exemplar in the representative class. At any given time, the minimum priority queue may only contain one node representing a given class [17, p. 420-421].

The priority queue is seeded with a representative example of each class. If there exists 10 classes, each containing 30 exemplars the priority queue will only contain a maximum of 10 nodes. The A^* algorithm is used to select and remove the minimum node from the priority queue. The minimum node will be expanded by computing the score of the exemplar which is pointed to by the iterator, $n+1$. The iterator value is then incremented to point to the next exemplar and the node's score is updated according to Equation 3.3, where:

$$\begin{aligned} g(n) =& \quad n_{score} + c(n+1) \\ h(n) =& \quad e * c(n+1) * d(n, N_{end}) \\ f(n) =& \quad g(n) + h(n) \end{aligned} \tag{3.3}$$

- e is a heuristic estimate value between $(0 < e < 1.0)$
- c is a function which computes the score between the current sample and exemplar
- d is a function which computes the distance between n and the last node in the class N_{end}

The expanded node is then enqueued in the priority queue and the expansion process repeats. The expansion process is terminated when $n = N_{end}$. The node which reaches the termination condition is determined to represent the best neighborhood for the samples' membership. Therefore, the sample is classified as being a member of the class represented by the terminating node. From this conclusion, it is important to note that each of the classes are assumed to contain exactly equal numbers of exemplars.

Given that a value for e is selected to ensure that $h(n)$ is admissible, then the A^* Nearest Neighborhood Classifier will determine the best possible class membership for the given sample [15, p. 95]. In the worst case, the algorithm will expand all possible nodes as defined by Equation 3.4, where C is the number of classes, and n is the number of exemplars in each class (note: all classes must contain an equal number of exemplars). In the best case, the algorithm will expand one exemplar from each class and immediately determine the best class leading to only $n-1$ more expansions (as shown in Equation 3.5).

$$O(n) = n(C-1) + 1 \rightarrow n \tag{3.4}$$

$$\Omega(n) = C + n - 1 \rightarrow n \tag{3.5}$$

As shown in Equation 3.4 and Equation 3.5, the growth of the algorithm is linear and bound by n. Given that the best and worst case for the proposed algorithm is bounded by n, Theta can be determined are shown in Equation 3.6.

$$\Theta(n) = n \tag{3.6}$$

The proposed algorithm requires that each representative class contain exactly the same number of exemplars. Care must be taken to ensure that the exemplars provide for the class are good representation of the gesture. All gestures to be recognised must also be distinctly separable into specific classes. The gesture recognition problem defined within this work meets all of the algorithm pre-conditions.

CHAPTER 4

Implementation

The implementation details of the re-designed gesture recognition software are discussed in this section. Beginning with Section 4.1, the details of how the software's runtime performance was increased are discussed. Section 4.2 covers the enhanced features used for the native implementation of the gesture recognition software. The relative distance function is explored in Section 4.3. In Section 4.4, the normalization between the distance function and the τ functions and the related balancing between those functions is explained.

Two new τ functions were implemented for the experiments conducted during the course of this work. The Zero reference τ function is described in Section 4.5. The details of the implementation of a new τ function introduced in Section 3.4 is explained in Section 4.6.

Additionally, two new classifiers were also implemented in order to conduct the experiments described within context of this work. The Nearest Neighborhood classifier was developed as a reference classifier for use as the basis of comparison for the results provided by the A^* Nearest Neighborhood classifier. The implementation of the Nearest Neighborhood classifier is examined in Section 4.7. The A^* Nearest Neighborhood classifier's implementation and pseudo code algorithm are presented in Section 4.8.

4.1 Runtime Performance

A native implementation of the DSTW Gesture Recognition system was developed to overcome the performance issues of the original Matlab version. C++ was selected as the primary language for the native implementation. OpenCV version 2.3.1 [18] was selected as the implementation of several computer vision algorithms. Additionally, the Boost libraries, version 1.46.1 [19], was selected to provide multi-threading, program configuration, and serialization functionality. The resulting code base was compiled using GNU g++ version 4.2.1 on both SuSE enterprise Linux and Mac OS X 10.6.8. The implementation of the DSTW algorithm was designed as a standalone library which accepts functor objects or function pointers to provide the distance and τ functions. The C++ implementation was used for all of the reported experiments.

4.2 Improved Features for Gesture Recognition

The original features were extended to include a bounding box around the subject's face. The center of the bounding box provides an origin for the reference coordinate system of positions within the image. The face detection algorithm was taken from the example of the Haar feature cascade classifier class defined in the OpenCV library based on the work presented in Viola and Jones [20]. The cascade classifier was set to return the largest object within the scene and the largest object returned was taken as the location of the subject's face. Additionally, a center point of the bounding box was added to the feature set to simplify the programing.

In addition to the face bounding box, a template image of the located hand region can also be added to the feature set. The hand region chip will be a gray scale, floating-point ($0 \rightarrow 1.0$) image with the mean value subtracted from each pixel.

The resulting hand region template was used to support the Template Matching τ Function (see Section 3.4).

The fundamental values of the vectors from which the distance function operates is still defined as a four-dimensional vector composed as (x, y, u, v) where (x, y) represents the hand position within the coordinate system, and (u, v) represents the hand velocity. The hand velocity is determined by the Farneback [21] optical flow algorithm, as implemented in the OpenCV 2.3.1 library [18]. Using the OpenCV implementation, the optical flow velocity was calculated over the entire image. Only the velocity values at the localized hand position were used as the (u, v) components.

4.3 Relative Distance Function

With support from the addition of the face location in the feature, two variations of each distance function were developed for comparison: absolute and relative. In the absolute versions, the units of the position component in the feature vector, (x, y), are measured in whole pixels, from the origin of the image frame $(0, 0)$. In the relative version, the units of the position component in the feature vector, (x, y), are also measured in whole pixels from the origin defined at the face location. In summary, absolute distance functions measure the absolute pixel positions within the image space, whereas relative distance functions measure everything from the center point of the face bounding box.

4.4 Distance and τ Normalization

In order to obtain the maximum performance from the distance and τ functions, the range of the functions must be balanced. Without balancing the functions, either the distance or the τ function will primarily determine the behavior of the DSTW algorithm. In order to address this issue, the range values will be normalized to the range of $[0 \rightarrow 1.0]$.

In the case of the distance function, the value of x is normalized with respect to the image width and the y value is normalized with respect to the image height. Using this method, the value $(0, 0)$ represents the upper-left corner while $(1.0, 1.0)$ corresponds to the lower-right corner.

For the τ function, the range values should also be normalized to the range of $[0 \rightarrow 1.0]$ to allow the sum of the distance and τ functions to be balanced. Therefore, the τ value will be normalized such that 0 is a no-cost transition and 1.0 is the most expensive transition.

4.5 Zero τ Function

The Zero τ function was developed as a reference implementation for a constant transition function for the DSTW algorithm. In this case, the function will return a 0 value for all pairs of (ω, ω') presented to the function (see Equation 4.1). In other words, all transitions have a fixed cost of zero. The overall effect of this is to drive the dynamic programing table only by the provided distance function. The results obtained from this implementation are used as a baseline for comparison with the Template Matching τ function.

$$\tau(\omega, \omega') = 0 \tag{4.1}$$

4.6 Template Matching τ Function

With the feature set now containing an image chip of a hand region, the results from a template matching function may now be used as part of the τ function. The goal is to allow an object which appears to be consistent between frames to produce a smaller transition cost than objects which are inconsistent using cross-correlation [22]. The cross-correlation operator can be used to determine the similarity of two consecutive image templates [16, p. 169]. The OpenCV `cv::matchTemplate` function using the method type of `CV_TM_CCORR_NORMED` provided the cross-correlation implementation.

Given a sufficient frame rate and that the image chips of the hand region are of the same object, then the cross-correlation score will be near 1.0. If the image chips are mismatched, then the cross-correlation score will be near zero. The result of the cross-correlation operation is then subtracted from 1.0 to produce a value which can be treated as a probability of match between the hand regions (see Equation 4.2[1]). The resulting value is then added to the transition cost; effectively making transitions between inconsistent chips more expensive than those between nearly consistent image chips.

$$\tau(\omega, \omega') = |\omega_i - \omega'_i| + |\omega_j - \omega'_j| + \big(1.0 - max(0, w_{image} \star w'_{image})\big) \tag{4.2}$$

4.7 Nearest Neighborhood Classifier

The Nearest Neighborhood classifier attempts to locate the class of a query by finding the best matching class. The best matching class is determined by computing a similarity score between the sample and every member exemplar of each given class. The score from each member is summed and the average score is recorded. The class with the lowest overall average score is selected as the sample's class. In this case, the membership of a given class determines the neighborhood and new members are expected to be near the candidate neighborhood.

[1]Note: The $\star$ symbol denotes the cross-correlation operator.

The nearest neighborhood classifier is an attempt to reduce the effect of a single, poorly selected exemplar causing mis-classifications. An implementation of the nearest neighborhood classifier was developed to act as a reference for the A^* Nearest Neighborhood classifier described in Section 3.5.

4.8 A^* Nearest Neighborhood Classifier

Based on the concepts introduced in Section 3.5, an implementation of the A^* Nearest Neighborhood classifier was developed according to the pseudo code shown in Algorithm 1. The ClassSet input value is a set of lists where each list represents a specific class to be identified, and the list is composed of n number of exemplars. The sample to be classified is represented as sample within the pseudo code. An estimate constant is shown as estimate provides a measure to estimate the similarity between the sample and the remaining exemplars in the class. A function which computes the similarity between the sample and the *exemplar* is shown as `similarity`. An additional function `distance` is provided to compute the distance between the current exemplar's position within the list and the last element of the list.

```
input  : A set of classes ClassSet each containing $n$ exemplars, a sample
         sample to be classified, a heuristic estimate constant estimate, a
         similarity function similarity, a distance function distance.
output: The nearest neighborhood class for sample contained within the
         set ClassSet

Initialization
minPriorityQueue = $\phi$
foreach class in ClassSet do
    $node_{class}$ = class
    $node_{key}$ = similarity (sample, $class_{exemplar[0]}$)
    $node_{gValue}$ = $node_{key}$
    $node_{iterator}$ = $class_{exemplar[1]}$
    $node_{end}$ = $class_{exemplar[n]}$
    enqueue (minPriorityQueue, node)
end

Iteration
node = dequeue (minPriorityQueue)
while $node_{iterator} \neq node_{end}$ do
    score = similarity (sample, $node_{iterator}$)
    $node_{gValue}$ = $node_{gValue}$ + score
    $node_{key}$ = $node_{gValue}$ + ( estimate $\times$ score $\times$ distance ($node_{iterator}$,
    $node_{end}$) )
    $node_{iterator}$ = $node_{iterator}$ +1
    enqueue (minPriorityQueue, node)
    node = dequeue (minPriorityQueue)
end
return $node_{class}$
```

Algorithm 1: A^* Nearest Neighborhood Classifier Algorithm

An implementation of Algorithm 1 was written in C++ for use in performing the experiments documented in this work. The ClassSet used in the experiments was the Palm Graffiti (see Figure 5.1) digit gestures [3]. The exemplars for each class were computed from a training set of data based on 10 individuals performing each gesture in the set 3 times (see Section 5.2). The gesture to be classified was provided as sample. Through experimental selection, a value of 0.825 was chosen for the estimate. The estimate values provide a good balance between acceptable positive detection and the number of nodes expanded during the A^* algorithm. In the context of this work, the DSTW algorithm was used as the `similarity` function. The Standard Template Library `std::distance()` function was used to calculate the distance between the current iterator and the end of the exemplar set.

CHAPTER 5

Summary Results and Analysis

A summary of the experimental results and an analysis of those results are presented within this section. Section 5.1 provides a summary of the reduction in terms of runtime that the enhanced software obtained. In Section 5.2, the two datasets that were used in the experiments as well as the training dataset are explained. The relative distance function's performance is discussed in Section 5.3. The performance of the Template Matching τ classifier is compared to the reference Nearest Neighborhood classifier in Section 5.4. Section 5.5 explores the performance of the A^* Nearest Neighborhood classifier utilizing a conservative and an aggressive heuristic function. Section 5.6, entitled "Effect of Multiple Candidates," discusses the performance impact of using multiple candidates within the composed query signals.

5.1 Runtime Performance

In discussing runtime performance in terms of wall time, a reference computing environment must be declared and used for the basis of comparison. In this section, the reference hardware and the experimental hardware is given in Table 5.1.

The performance limitations of Matlab presented a significant restriction in the number of experiments that could be performed within a reasonable amount of time. With a recognition database size of 10 classes, each containing 3 exemplar gestures, the Matlab implementation was only able to complete an average of one query per several minutes of execution time on reference hardware. After significant optimiza-

Table 5.1. Hardware Platforms Utilized for Development and Experiments

	Reference	**Experimental**
System:	Macintosh OS X 10.6.8	SuSE 11 Linux 3.0.58 SMP
Processor:	2.8GHz Intel Core 2 Duo	Quad 10-core 2.4GHz Intel Xeon
Memory:	4 GB	256 GB
Matlab:	R2008b	N/A
Tool chain:	g++ 4.2.1	g++ 4.3.4
Libraries:	Boost 1.46.1, OpenCV 2.3.1	

tions of the Matlab code, the performance was improved to an average execution time of 55 seconds per query (see Table 5.2).

The key contributing factor affecting the runtime execution performance is the selection of the distance and τ functions. Within the context of the following performance factors, the Manhattan function as previously introduced in 2.2.2 was used for the distance function. The Manhattan function as previously introduced in 2.2.3 was used for the τ functions. These functions were also implemented in the Matlab version of the gesture recognition software, and hence are used as the base function for comparison with the native software implementation.

Table 5.2. Runtime Performance for DSTW Gesture Recognition on Reference Hardware

Implementation	**Runtime per Query**
Matlab (original)	$\sim$2 minutes, 15 seconds
Matlab (optimized)	$\sim$55 seconds
Native Executable	$\leq$ 1 seconds

The native implementation proved to perform significantly better on the reference hardware than the Matlab implementation. During the experimentation phase of this work, it is worth noting that the native code base was multi-threaded to enable simultaneous queries to execute in parallel. The primary system used for the experiments was a Dell PowerEdge R950 with quad ten-core Intel Xeon 2.4GHz processors with 256GB RAM running SuSE Enterprise Linux server version 11, service pack 2, kernel 3.0.58 SMP (see Table 5.1). The experiments executed without intervention for several weeks. The native implementation proved to be very robust.

5.2 Datasets Used in Experiments

In the following section, the datasets which were used during the experiments are examined. The same set of gestures that were used in the original Alon et al. [3] paper were also used for this work. The datasets contain individuals performing the digit gestures defined by the Palm Graffiti alphabet (see Figure 5.1 provided by `http://www.techdc.com/rip-original-palm-os`). The dataset was segmented into Training, Easy, and Hard sets. Each of the datasets included ground-truth files which enabled automatic scoring. The datasets were delivered and processed as AVI video files, captured at 320x240 resolution with a 24 bit RGB color space.

5.2.1 Training

The Training dataset provided 10 classes (digits 0-9), each containing 30 exemplars. The resulting Training dataset was composed into the recognition database and used during all experiments. The training videos were produced by 10 different individuals, each performing the gesture for each digit three times. The training gestures were made with a green gloved hand to enable the software to detect a specific color for improved recognition. An example image from a Training dataset video is shown in Figure 5.2. The frame rate for the Training dataset was 30 Hz.

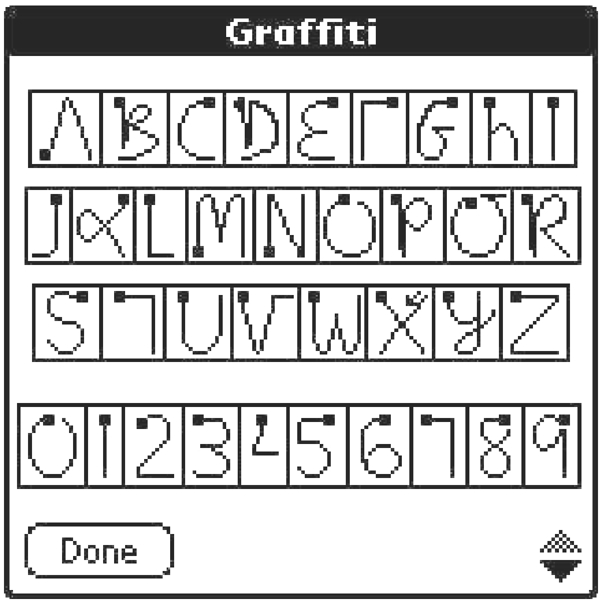

Figure 5.1. Palm Graffiti Alphabet [1].

5.2.2 Easy Query

The Easy dataset was used as the source of the easy queries. The Easy dataset is characterized by having a single subject performing the gesture, with little or no background movers. Additionally, effects of sun-busts within the video image have been minimized. A representative image from the dataset is shown in Figure 5.3. The Easy dataset was composed by the same 10 individuals that performed the gestures in the Training dataset. In the Easy dataset, each person performed each digit three times creating a dataset containing 300 query gestures. The frame rate for the Easy dataset was 30 Hz. Test 59 obtained the best results on the Easy dataset (see 5.3).

5.2.3 Hard Query

The Hard dataset was used as the source of the hard queries. The Hard dataset is characterized by having multiple subjects within the scene. The gesturing subject

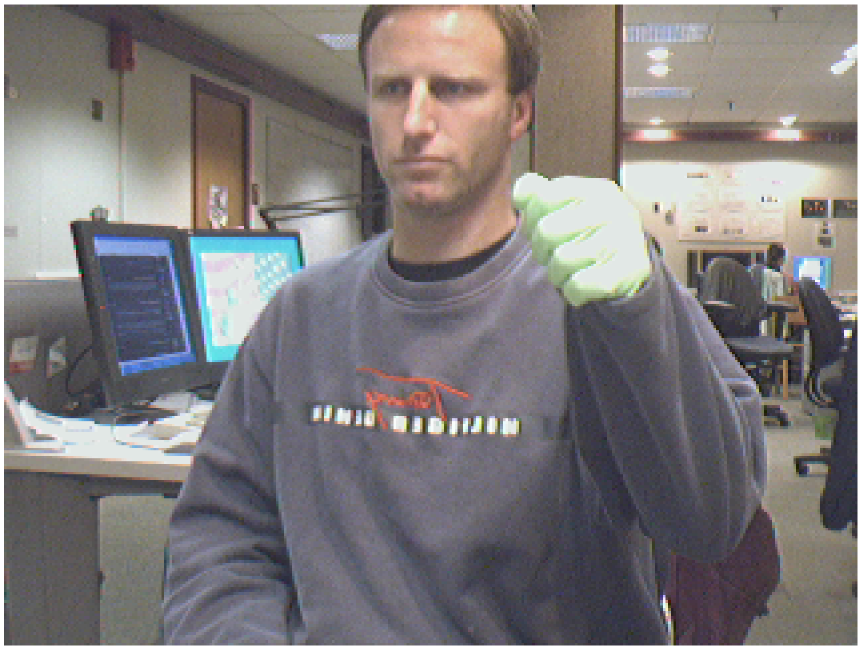

Figure 5.2. An example image taken from the Training dataset.

is often positioned to one side of the image while other participates are moving in the background. Often the background subjects turn to face the camera, as shown in Figure 5.4. The Hard dataset was composed by 7 individuals that performed each gesture twice, creating a dataset containing 140 queries. The frame rate for the Training dataset was 15 Hz. Test 154 obtained the best results on the Hard dataset (see 5.4).

5.3 Relative Distance Function

For the experiments that were executed on the Easy dataset, the relative distance function generally seems to have had no significant improvement on the recognition performance of the system (see Figures 5.5 and 5.6). The figures clearly show that the most significant impact to recognition performance is the number of candi-

Table 5.3. Easy dataset best result (Test 59, 100.0%)

Confusion Matrix										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	30	0	0	0	0	0	0	0	0	0
1	0	30	0	0	0	0	0	0	0	0
2	0	0	30	0	0	0	0	0	0	0
3	0	0	0	30	0	0	0	0	0	0
4	0	0	0	0	30	0	0	0	0	0
5	0	0	0	0	0	30	0	0	0	0
6	0	0	0	0	0	0	30	0	0	0
7	0	0	0	0	0	0	0	30	0	0
8	0	0	0	0	0	0	0	0	30	0
9	0	0	0	0	0	0	0	0	0	30

dates presented to the DSTW algorithm. In fact, the relative verses absolute distance functions produced nearly identical results.

The Hard dataset presented additional challenges. As a result of presenting an additional face to the camera in the Hard dataset, the face detection algorithm would become confused and report the non-gesturing face as the face center point. As discussed in Section 3.3, a misclassified face center point location could cause a reduction in recognition performance. Experiments on the Hard dataset prove this to be the case. In order to address this problem, a more sophisticated face detector and tracker should be used. Once the face location of the subject who is gesturing is determined, that face should be tracked between frames to ensure a relatively stable origin point for the gesturing space.

In conclusion, the experiments performed within this work show that there is not enough meaningful data to suggest using a relative distance function over using an absolute function. However, when executed with an early development dataset that was not included in the final experiments, the relative distance function performed considerably better. In a dataset in which the subject performing the

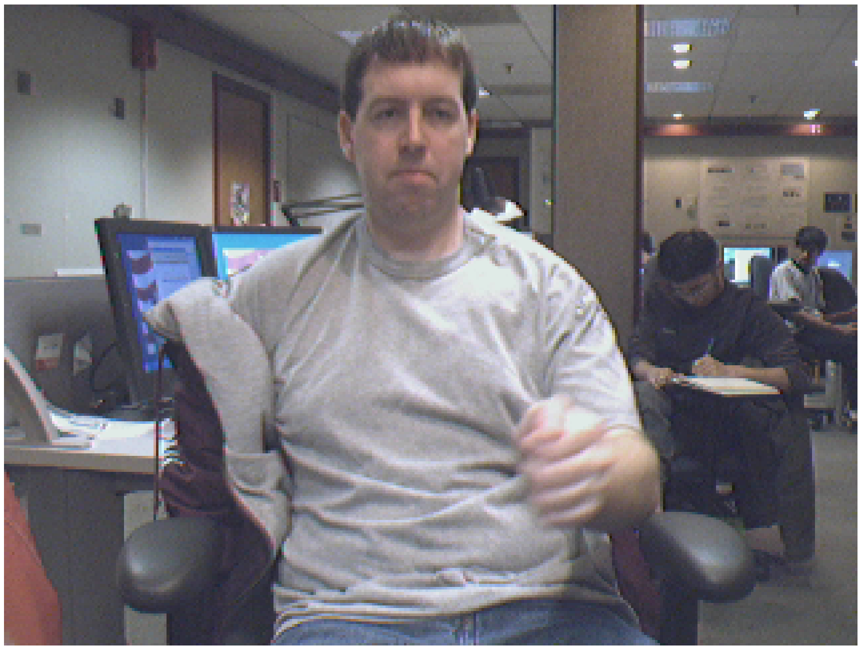

Figure 5.3. An example image taken from the Easy dataset.

gesture is in motion, the relative distance function may prove to be useful. Lacking experimental data to suggest that using a relative distance function would reduce recognition performance in any meaningful way, experiments performed within the remainder of this work are based on the results of relative distance functions.

5.4 Template Match τ Function

In the following sections, the performance results from the experiments isolating the effects of the Template Matching τ functions are discussed. For all of the experiments covered in this section, the Relative Manhattan Normalized distance function and the Nearest Neighbor search method were used to produce the results. In Section 5.4.1, impact on the positive recognition rates between the Template Matching, Zero, and Manhattan Normalized τ functions when using the Easy dataset queries is discussed. Section 5.4.2 covers the performance of the three functions when the

Table 5.4. Hard dataset best result (Test 154, 76.4%)

Confusion Matrix										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	8	1	0	0	4	0	1	0	0	0
1	0	13	0	0	0	0	0	1	0	0
2	0	0	12	0	1	0	0	1	0	0
3	0	1	0	13	0	0	0	0	0	0
4	0	0	0	0	14	0	0	0	0	0
5	0	0	0	0	3	9	0	2	0	0
6	1	0	0	0	3	0	10	0	0	0
7	0	6	0	0	0	0	0	8	0	0
8	0	0	1	1	0	1	0	0	11	0
9	0	1	0	0	1	1	0	2	0	9

Hard dataset is used for the queries. The runtime impact of the Template Matching τ function is covered in Section 5.4.3. A conclusion is then presented in Section 5.4.4.

5.4.1 Results from the Easy Dataset

On the Easy dataset, the Template Match τ function performed equal to, or better than both Zero and Manhattan Normalized functions (see Figure 5.7). When used with a single candidate, all three τ functions were able to obtain a result of 100% correct recognition of the presented queries. All three functions demonstrated a reduction in recognition performance when the number of candidates increased. With 5 candidates under consideration, all three functions were able to correctly recognize 99.67% of the presented queries. The data shows that once the number of candidates is increased to 10 or more, both the Zero and Manhattan Normalized functions' recognition performance continue to drop. The Template Matching function's recognition performance levels off at the 99.67% mark. This behavior suggests that the Template Matching function is at least maintaining the level of performance when presented with an increasing opportunities for error.

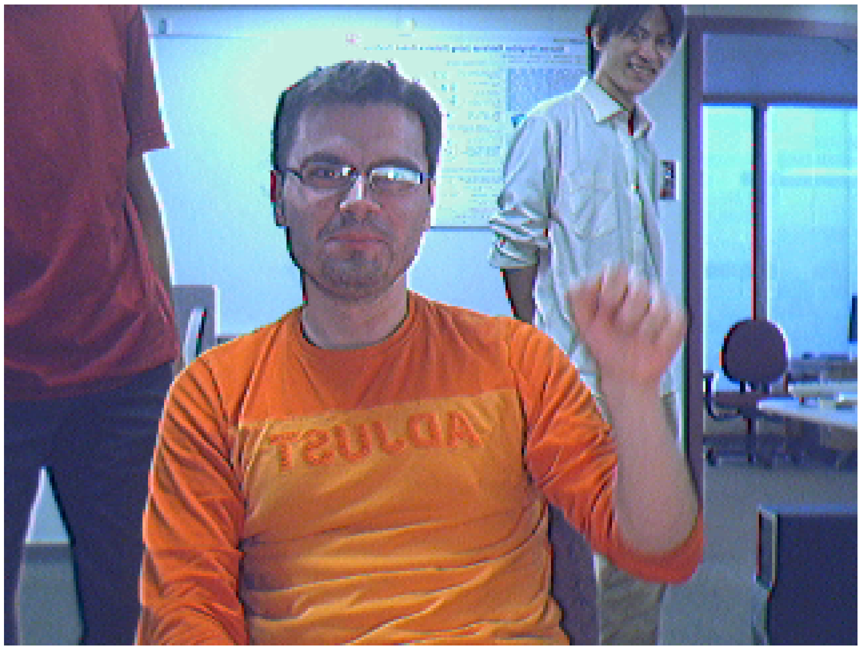

Figure 5.4. An example image taken from the Hard dataset.

5.4.2 Results from the Hard Dataset

The results from the Hard dataset are quite different and shown in Figure 5.8. All three τ functions are able to correctly recognize 53.57% when presented with a single candidate. When presented with 5 candidates, both the Zero and the Manhattan Normalized functions are able to increase gesture recognition to 56.42%. However, the Template Matching function's performance drops to 50%. When presented with 10 candidates, the Template Matching function increases slightly to 50.67% while the Zero and Manhattan Normalized functions continue to fall to 37.86%. Given the length of runtime needed to complete the experiments using the Template Matching function, data for a comparison at the 15 and 20 candidate level is not available. It would be interesting to compare those results to determine if the general trend continues.

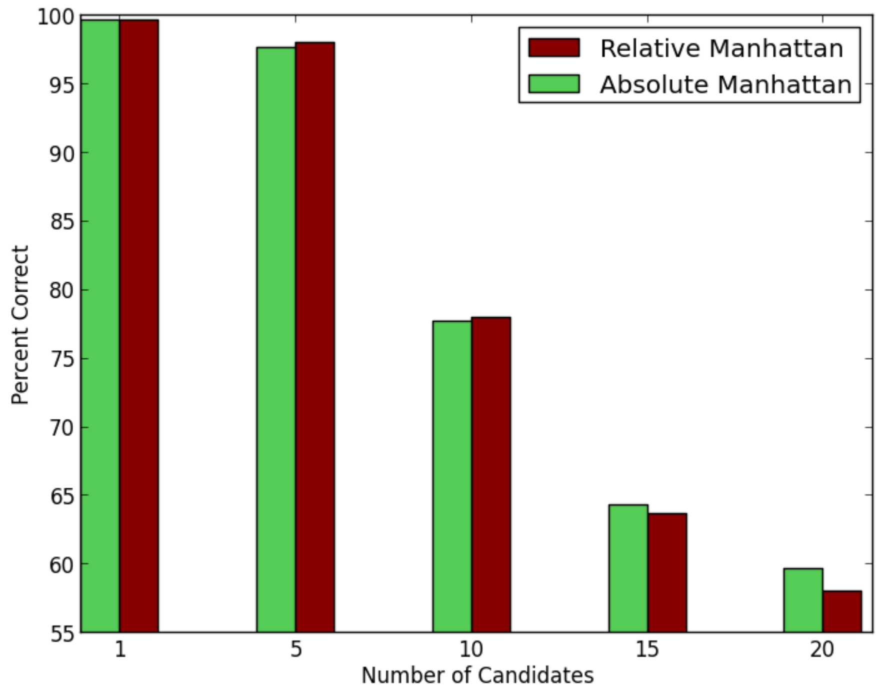

Figure 5.5. Comparison between Relative and Absolute Manhattan Distance functions. (τ: Manhattan, Search: Nearest Neighbor, Dataset: Easy).

5.4.3 Impact on Runtime Performance

Experiments on both datasets seem to indicate that the Template Matching τ function may offer some performance improvement when numerous candidates are generated for a given query. However, the major drawback to using the Template Matching function is the significantly increased runtime as shown in Figure 5.9. In the event that the runtime can be reduced to a point where it becomes feasible to run a query with 50, 75, and 100 candidates, then the overall recommendation to avoid the Template Matching τ algorithm would need to be revisited.

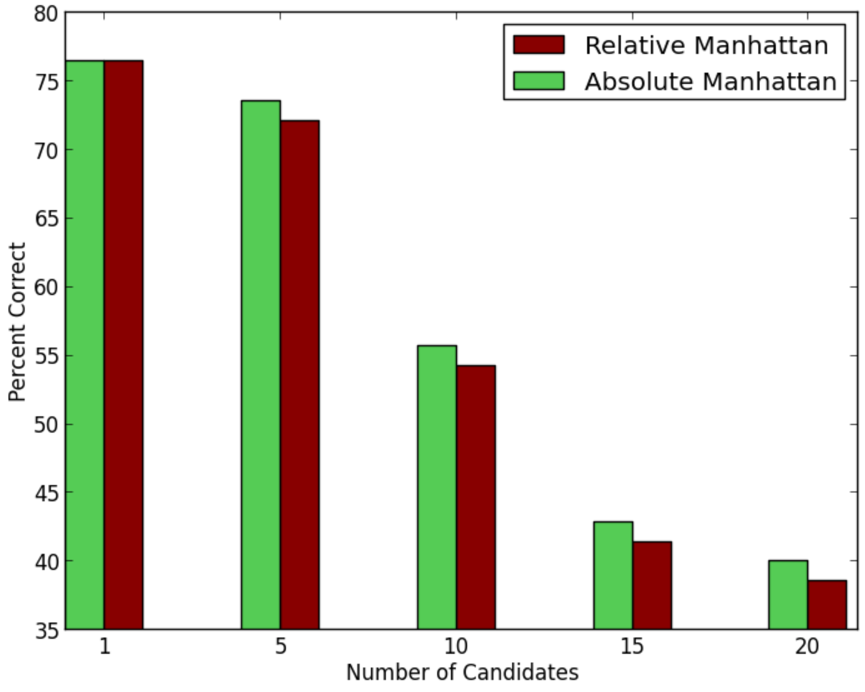

Figure 5.6. Comparison between Relative and Absolute Manhattan Distance functions. (τ: Manhattan, Search: Nearest Neighborhood, Dataset: Hard).

5.4.4 Conclusion

In conclusion, the Template Matching τ function does not add sufficient value to the gesture recognition problem to warrant the additional runtime. The clear tendency of the function to stabilize when presented with additional candidates indicates that the Template Matching τ function was influencing the transition table used in the DSTW algorithm. For all of the experiments in which the Template Matching τ function was selected, it should be noted that the distance and τ functions were balanced to equally influence the DSTW matrix. It is possible that additional experimentation using different weighting might provide useful results.

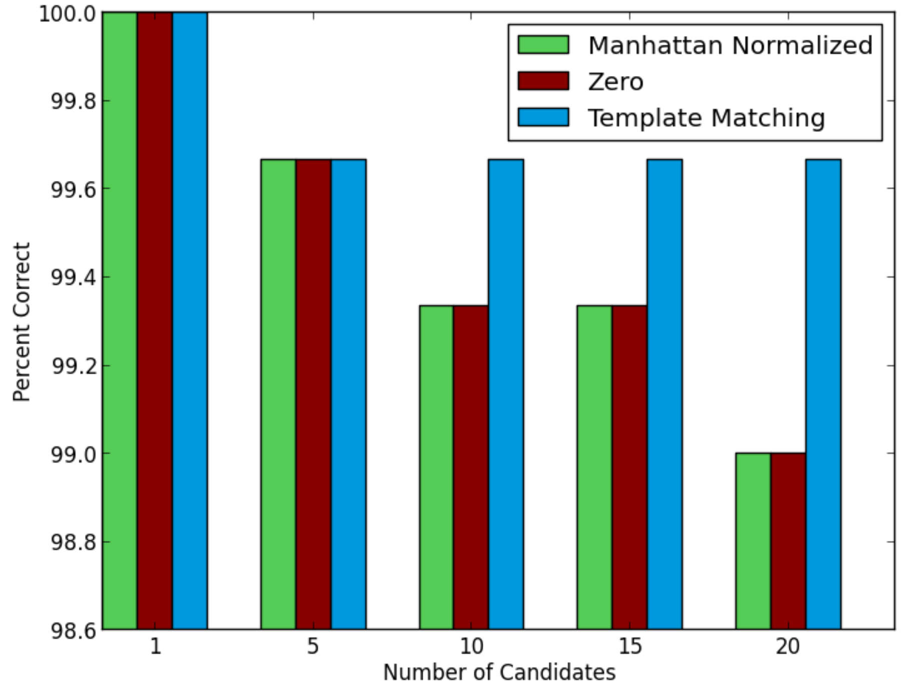

Figure 5.7. Comparison of the Manhattan Normalized vs. Zero vs. Template Matching τ functions effective positive recognition. (Distance: Relative Manhattan Normalized, Search: Nearest Neighbor, Dataset: Easy).

5.5 A^* Nearest Neighborhood Classifier

The performance of the A^* Nearest Neighborhood classifier is discussed in following sections. All experiments in the subsequent sections utilized the Relative Manhattan distance function and Manhattan Normalized transition function as τ. The experiments were conducted using a conservative and aggressive version of an inadmissible heuristic function. In Section 5.5.1, experimental results on the effect of the classifier on recognition performance when using the Easy dataset is presented and analyzed. The A^* Nearest Neighborhood classifier effect on gesture recognition performance from the Hard dataset is discussed in Section 5.5.2. The impact to runtime

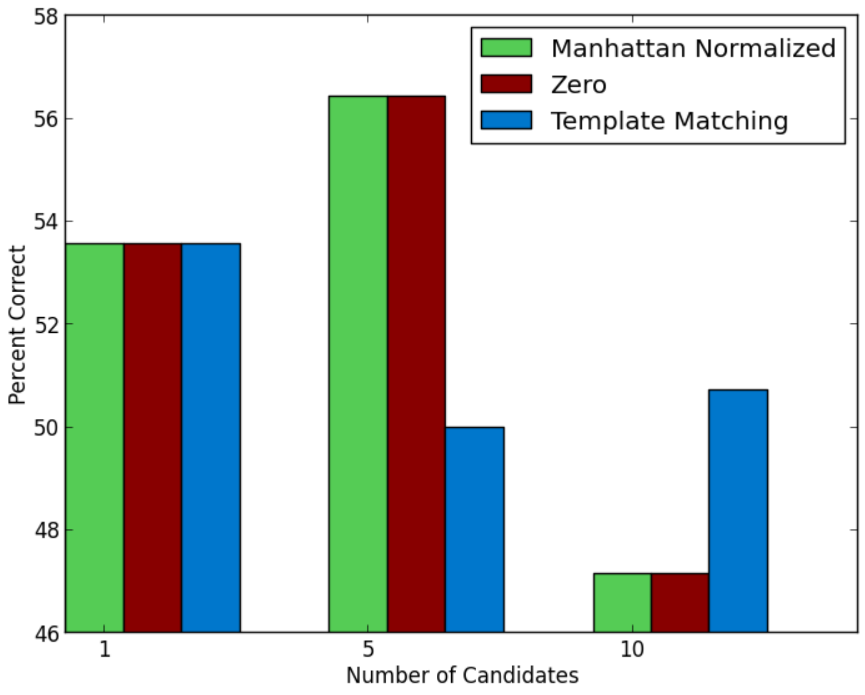

Figure 5.8. Comparison of the Manhattan Normalized vs. Zero vs. Template Matching τ functions effective positive recognition. (Distance: Relative Manhattan Normalized, Search: Nearest Neighbor, Dataset: Hard).

performance is covered in Section 5.5.3. A conclusion based upon the experimental data is presented in Section 5.5.4.

5.5.1 Performance on the Easy Dataset

On the Easy dataset, the impact to positive gesture recognition from the A^* Nearest Neighborhood classifier must be compared against the Nearest Neighborhood classifier. Both methods are attempting to classify the given sample by computing the average score of the entire class membership and select the class with the lowest score as the sample class. In Figure 5.10, the results from the Nearest Neighbor classifier are provided for reference. As Figure 5.10 shows, the results from both the conservative

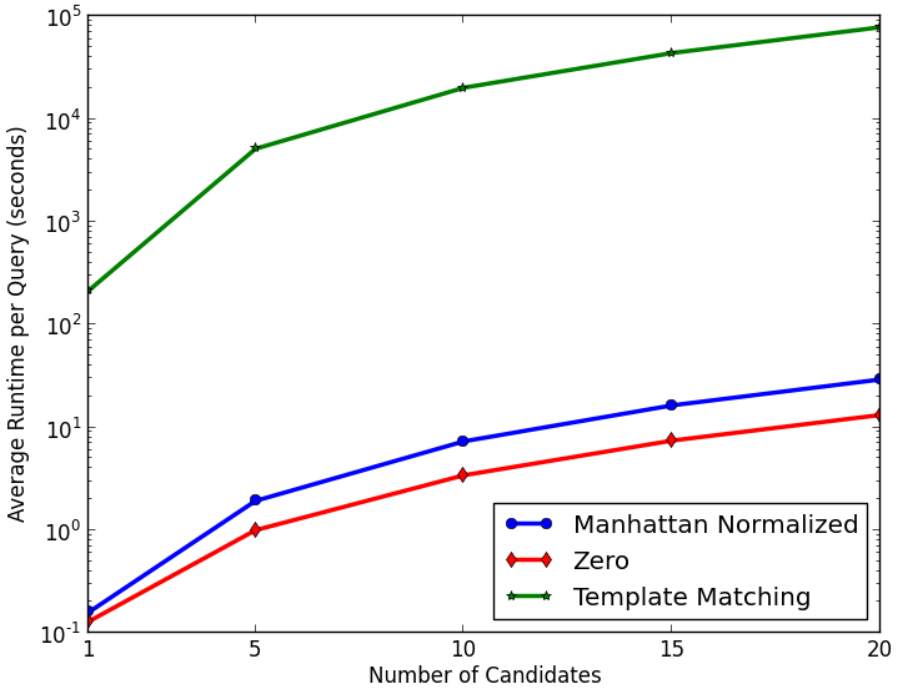

Figure 5.9. Comparison of the Manhattan Normalized vs. Zero vs. Template Matching τ functions effective runtime. (Distance: Relative Manhattan Normalized, Search: Nearest Neighbor, Dataset: Easy).

and aggressive heuristic functions perform identically, with the exception of the tests executed with 10 candidates. In the 10 candidate case, the aggressive heuristic failed to correctly classify one query that the conservative heuristic classified correctly.

By choosing an aggressive heuristic for the A^* algorithm, a trade-off between correctness and speed is made. It should be noted, however, that the performance from the Nearest Neighborhood classifier also mis-classified one query. The Nearest Neighborhood classifier computes results for every member of the class and therefore should provide the optimal performance for classifiers of this type. It is possible that the conservative A^* version produced a false positive classification rather than the admissibility of the heuristic function causing the error.

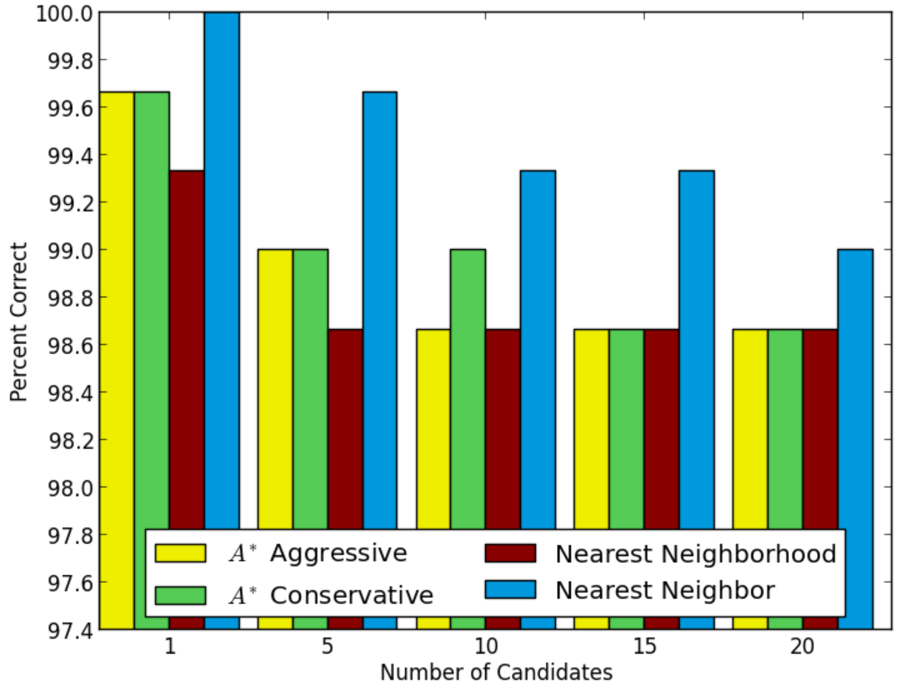

Figure 5.10. Comparison of the A^* Nearest Neighborhood vs. Nearest Neighborhood vs. Nearest Neighbor classifiers. (Distance: Relative Manhattan Normalized, τ: Manhattan Normalized, Dataset: Easy).

5.5.2 Performance on the Hard Dataset

Figure 5.11 shows a comparison of the A^* Nearest Neighborhood, Nearest Neighborhood, and Nearest Neighbor classifiers. With the Hard dataset, the aggressive A^* heuristic was used for all experiments. With this dataset, the results show a significant improvement in performance for the A^* classifier. In all cases except the 5 candidate case, A^* provides better recognition performance than both Nearest Neighborhood and Nearest Neighbor. However, Nearest Neighborhood is expected to equal or out perform A^*. One explanation for the presented results is that the A^* algorithm did not expand the nodes which provided closer matches to the incorrect classes. Since A^* classifier is traversing to the last exemplar defined within the class, the algorithm may reach the terminal condition prior to expanding nodes which could end in false classifications. In the 5 candidate case, the difference was not considered meaningful.

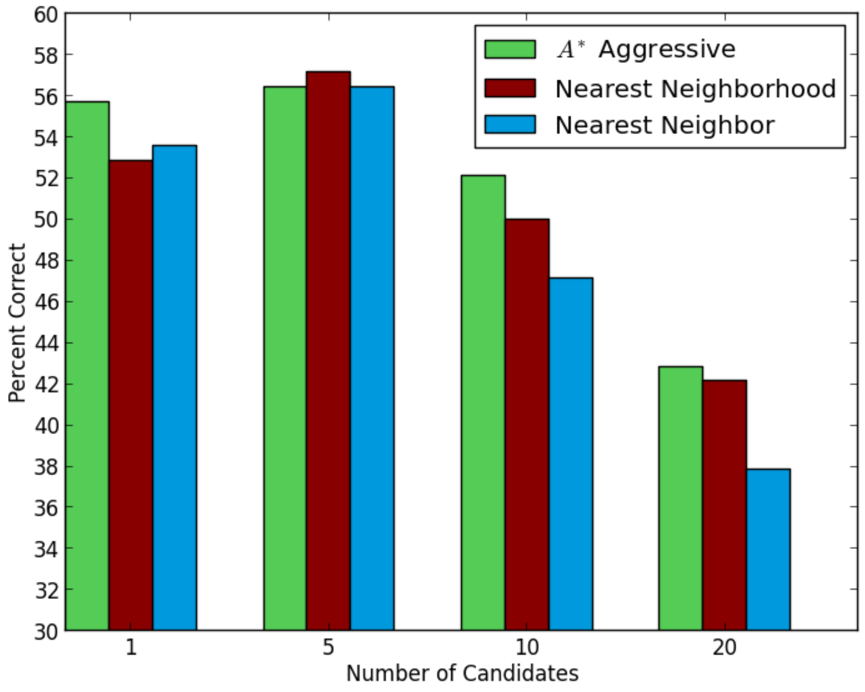

Figure 5.11. Comparison of the A^* Nearest Neighborhood vs. Nearest Neighborhood vs. Nearest Neighbor classifiers. (Distance: Relative Manhattan Normalized, τ: Manhattan Normalized, Dataset: Hard).

5.5.3 Effect on Runtime Performance

As expected of an A^* family algorithm, runtime was significantly reduced in comparison to the performance of the other classifiers. The runtime results for the Easy dataset are shown in Figure 5.12. As the number of candidates increase, the graph of the A^* Nearest Neighborhood classifier can be seen falling away for the computation time needed by both Nearest Neighbor and Nearest Neighborhood classifiers. The lowest line within the graph represents the effects on runtime of selecting an aggressive heuristic for the A^* algorithm. Also as expected, the aggressive heuristic further reduced runtime. The impact to the runtime for the Hard dataset is shown in Figure 5.13.

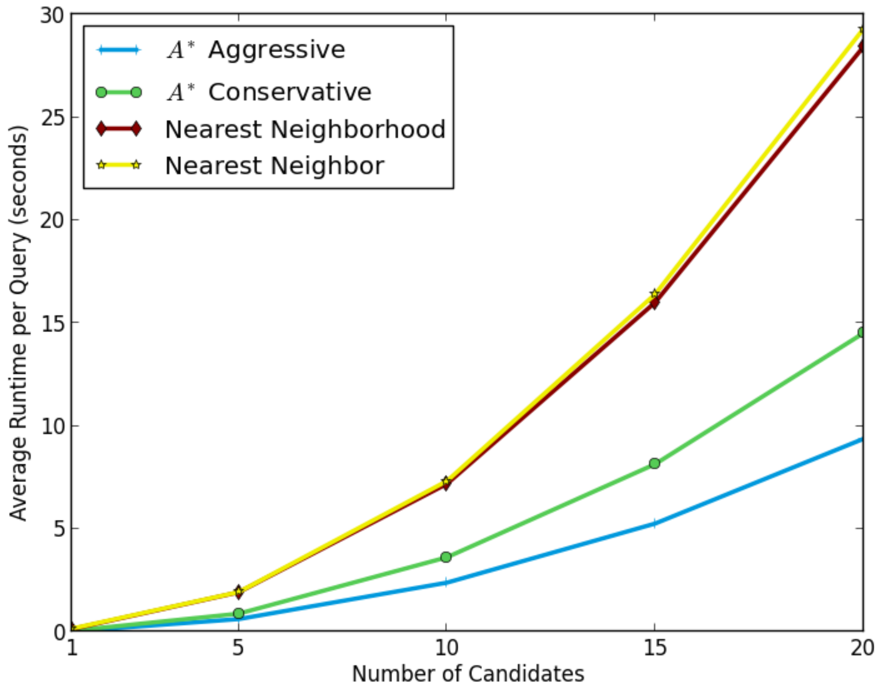

Figure 5.12. Comparison of the A^* Nearest Neighborhood vs. Nearest Neighborhood vs. Nearest Neighbor classifiers. (Distance: Relative Manhattan Normalized, τ: Manhattan Normalized, Dataset: Easy).

In terms of experimental algorithm complexity performance, both the $O(n)$ (see equation 3.4) and $\Omega(n)$ (see equation 3.5) bounds are linear to the number of exemplars and classes as previously shown. The results from the experiments shows that the average number of expended nodes was 139.26. The average maximum nodes expanded was found to be 231.54 out of a maximum possible of 271 nodes as given

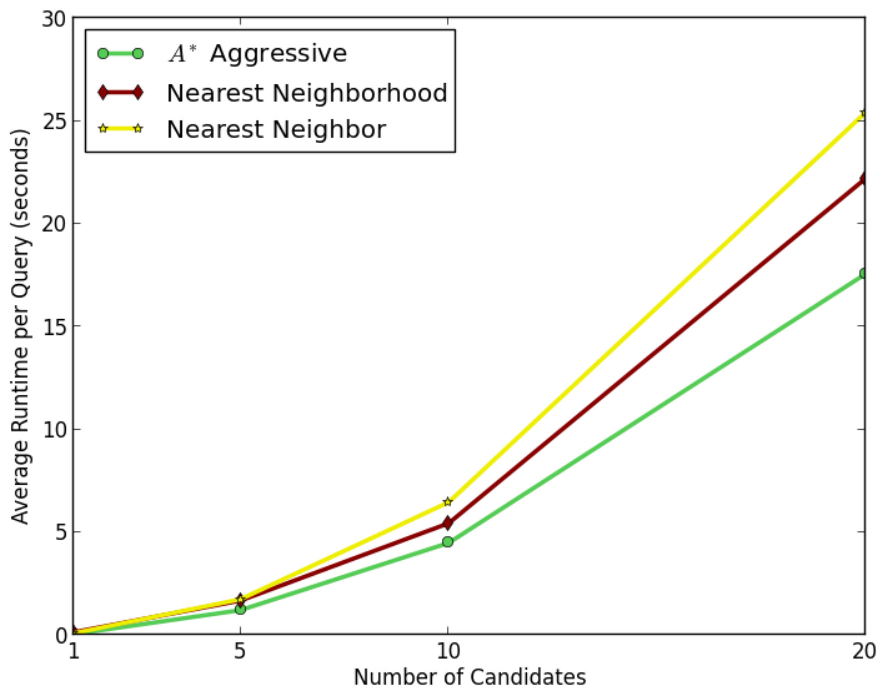

Figure 5.13. Comparison of the A^* Nearest Neighborhood vs. Nearest Neighborhood vs. Nearest Neighbor classifiers. (Distance: Relative Manhattan, τ: Manhattan Normalized, Dataset: Hard).

from $O(n)$ equations previously presented. Examining the lower bound experimental results, the average minimum expansion depth was found to be 70.06 nodes; above the theoretical lower bound of 39 nodes as given by the previously presented $\Omega(n)$ equation.

5.5.4 Conclusion

In conclusion, the A^* Nearest Neighborhood classifier does reduce runtime at the cost of positive recognition performance. When eliminating the Nearest Neighbor classifier performance from the evaluation, the A^* Nearest Neighborhood classifier performs on par with the ideal Nearest Neighborhood classifier; however the advantage

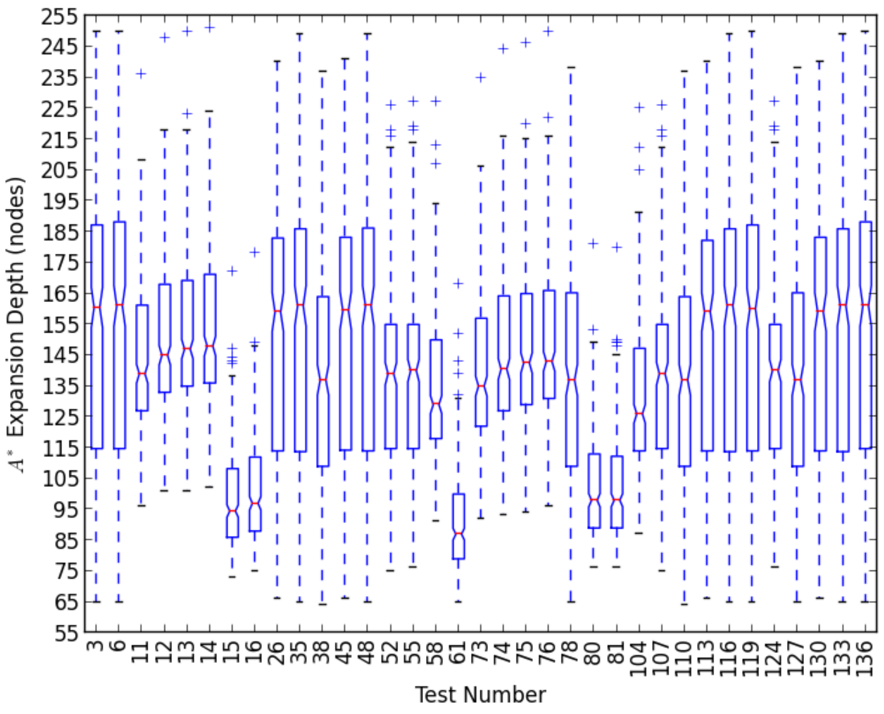

Figure 5.14. Summary of A^* minimum, average, and maximum search depths for all experiments using a conservative heuristic. (Distance: Relative Manhattan Normalized, τ: Manhattan Normalized, Dataset: Easy).

of speed clearly goes to the A^* Nearest Neighborhood classifier. The decision to use a conservative or an aggressive heuristic must be made with a clear understanding of the specific application's tolerance to incorrect results.

Another interesting conclusion from Figure 5.11; Nearest Neighbor is no longer the premier classifier providing the best recognition performance. Given that the same individuals that created the gestures for the Easy dataset also created the Training dataset, the performance recorded from Nearest Neighbor on the Easy dataset may be the result of over training. An interesting experiment would be to have a different set of individuals create an alternative easy query dataset for execution with the existing Training dataset.

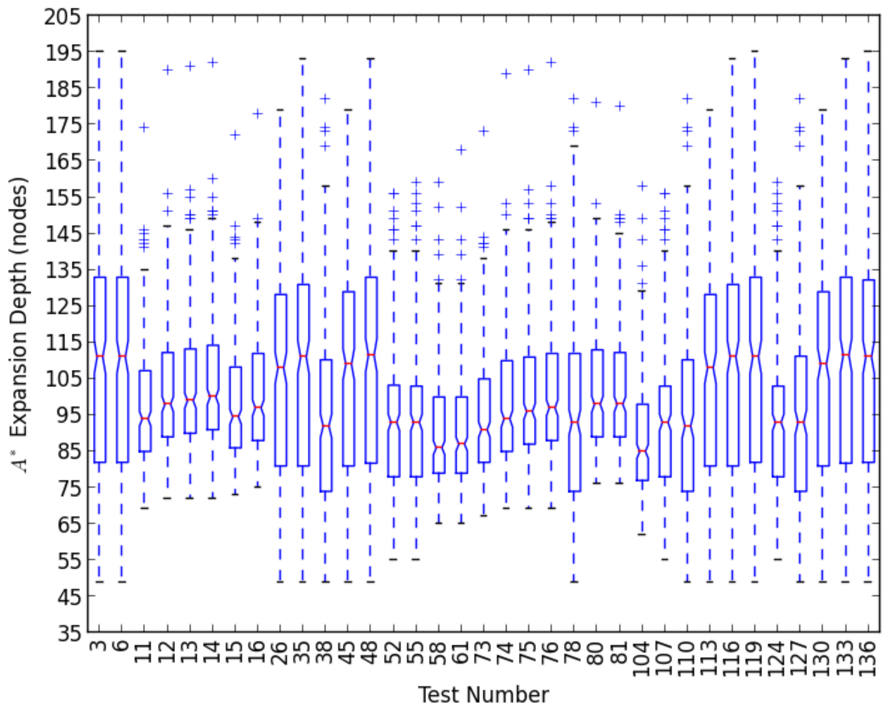

Figure 5.15. Summary of A^* minimum, average, and maximum search depths for all experiments using an aggressive heuristic. (Distance: Relative Manhattan Normalized, τ: Manhattan Normalized, Dataset: Easy).

5.6 Effect of Multiple Candidates

Throughout the experiments, the single most significant effect on the positive gesture recognition was the number of candidates. With the exception of two test cases within the Hard dataset, increasing the number of candidates reduced recognition performance. A contributing factor to this behavior may be explained by examining the detector. The detector written for use in this work closely followed the detector described in Alon et al. [3]. One key difference between the two implementations was in the hard coded hand size parameter. In the original detector, the hand size was fixed. In the detector written for this work, the hand size was computed in each frame as one-half the size of the detected face.

The DSTW algorithm was designed to produce the best possible alignments between sample and exemplar when a weak detector provides multiple candidates. In the experiments presented within this work, the additional candidates caused the opposite effect. It is possible that the rewritten detector simply performed better than the detector used the Alon et al. paper [3]. In the case of a strong detector, the first candidate may always be the hand to be tracked. By providing additional candidates, the DSTW is given opportunities to find a better alignment to the extraneous candidates than a slightly less optimal, but accurate, alignment to the true sample.

CHAPTER 6

Future Work

Within this section, suggestions for future work are presented. In Section 6.1 an experiment that could improve the results of the Template Matching τ is outlined. Section 6.2 describes an experiment to test the usability of the relative distance functions within a scene that contains a moving gesturing subject. An experiment to determine the ideal inadmissible heuristic is discussed in Section 6.3. An alternative τ function designed for a stable detector is presented in Section 6.4.

6.1 GPU Processing of the Template Matching Classifier

The largest disadvantage to using the Template Matching classifier was the negative impact to runtime. The Template Matching classifier could prove significantly more useful when provided with more candidates than were tested within this work. The limitation to increasing the number of candidates was constrained by the amount of time available to conduct experiments. The impact to runtime may be minimized by executing the cross-correlation operation on a high-performance graphics processing unit (GPU) or other specialized hardware.

Another possibility for the improvement to the Template Matching classifier is to identify another method of performing the template match that would be more computationally efficient than the cross-correlation operation chosen for this work. Additional experiments using the Template Matching classifier were initially performed using the square difference algorithm as implemented in the OpenCV library. The initial impact on runtime and recognition performance between square difference and cross-correlation where nearly identical and cross-correlation was selected for all future experiments.

6.2 Testing Relative Manhattan Distance Function with Motion

Given the results from testing the Relative distance function with both the Easy and Hard dataset, it is possible that the function would perform better when the scene was less stable. Given that the origin of the reference frame for matching the trajectories is based on the gesturing subject's face, should the subject move about the scene while still maintaining a reasonably clear view of the face, the relative function should provide significantly better results than the absolute function. In order to conduct this experiment, the face detector would need to be enhanced to allow for the tracking of the face within the scene.

6.3 Inadmissible Heuristics for A^* Nearest Neighborhood Classifier

As presented within the context of this work, the A^* Nearest Neighborhood classifier was executed using two different heuristic functions: conservative and aggressive. The aggressive function produced considerably larger estimates to the end node than the conservative function. However, no effort was made to determine the most ideal inadmissible heuristic for the gesture recognition problem. A series of experiments on the Hard dataset should be devised to determine an inadmissible heuristic with an ideal balance between accuracy and runtime.

6.4 A τ Function for a Stable Detector

The core concept presented here is that the detector generally produces detection in order of likelihood of being the correct signal, then the signal should rarely transition between candidate position. Therefore, any transition in the dynamic programing matrix that is farther than neighboring candidates should be weighted heavily. Based on the results presented within, a good additional experiment would be to use the candidate ranking as an input to the τ function. If a strong detector is able to consistently produce properly ranked results, then the position within the candidate vector would be a good indicator as to the validity of the transition. For example, if the ranking is stable, then a transition between non-adjacent candidates could have a larger weighted cost than between neighboring candidates.

APPENDIX A

Summary of Experimental Results

Table A.1. Abbreviation Table

Abbreviation	**Meaning**
	Datasets
E	Easy
H	Hard
	Distance Functions
M	Manhattan
R	Relative Manhattan
N	Relative Manhattan Normalized
	Tau (τ) Functions
M	Manhattan
N	Manhattan Normalized
T	Template Cross-Correlation Normalized
Z	Zero
	Classifiers
N	Nearest Neighbor
H	Nearest Neighborhood
A	A^* Nearest Neighborhood

Summaries of the raw results from all of the experiments discussed within the paper are included in this section. The values listed under ”Runtime” are in seconds and the minimum, average, and maximum values are presented. For experiments executed using the A^* Nearest Neighborhood classifier, the minimum, average, and maximum search depths (node expanded) are also provided with respect to the conservative heuristic. The abbreviations used in the tables are shown in Table A.1.

Table A.2. Summary of Experiments: Test 1 to 25.

Test #	**Dataset**	**Candidates**	**Distance Function**	**Tau (τ) Function**	**Classifier**	**A^* Depth (nodes)**			**Runtime (seconds)**			**% Correct**
						MIN	AVG	MAX	MIN	AVG	MAX	
01	E	20	M	M	N		N/A		3	16.5	32	59.7
02	E	20	M	M	H		N/A		3	16.0	33	83.0
03	E	20	M	M	A	65	153.9	250	0	8.4	25	76.0
04	E	20	R	M	N		N/A		4	15.7	34	58.0
05	E	20	R	M	H		N/A		3	16.1	33	82.7
06	E	20	R	M	A	65	154.5	250	0	8.4	24	75.3
07	E	5	N	N	H		N/A		0	1.9	4	98.7
08	E	10	N	N	H		N/A		2	7.2	14	98.7
09	E	15	N	N	H		N/A		3	16.0	32	98.7
10	E	20	N	N	H		N/A		6	28.4	54	98.7
11	E	5	N	N	A	96	143.8	236	0	0.9	2	99.0
12	E	10	N	N	A	101	149.6	248	0	3.6	8	99.0
13	E	15	N	N	A	101	151.9	250	1	8.2	18	98.7
14	E	20	N	N	A	102	153.1	251	2	14.5	32	98.7
15	E	5	N	T	A	73	97.8	172	178	1600.9	6755	99.0
16	E	10	N	T	A	75	100.6	178	798	6571.0	27547	99.0
17	E	5	N	T	H		N/A		1109	5140.8	15149	99.3
18	E	10	N	T	H		N/A		4257	20284.1	60465	98.7
19	E	5	N	T	N		N/A		1081	5134.0	15474	99.7
20	E	10	N	T	N		N/A		4117	20164.6	57714	99.7
21	E	10	M	M	N		N/A		1	4.1	8	77.7
22	H	10	N	T	N		N/A		4661	32054.2	189226	50.7
23	H	10	N	T	H		N/A		4676	32067.9	186746	52.9
24	H	10	N	T	A	78	158.4	240	981	16476.0	125641	53.6
25	E	10	M	M	H		N/A		0	4.1	9	88.7

Table A.3. Summary of Experiments: Test 26 to 50.

Test #	**Dataset**	**Candidates**	**Distance Function**	**Tau (τ) Function**	**Classifier**	**A^* Depth (nodes)**			**Runtime (seconds)**			**% Correct**
						MIN	**AVG**	**MAX**	**MIN**	**AVG**	**MAX**	
26	E	10	M	M	A	66	151.7	240	0	2.1	6	85.7
27	H	10	N	N	H		N/A		1	5.4	10	50.0
28	H	10	N	N	A	108	210.9	282	0	4.5	10	52.1
29	H	10	N	N	N		N/A		2	6.5	11	47.1
30	H	20	M	M	H		N/A		4	14.1	25	40.0
31	H	20	R	M	H		N/A		4	13.9	29	38.6
32	H	20	N	N	H		N/A		6	22.2	44	42.1
33	E	15	M	M	N		N/A		2	9.2	18	64.3
34	E	15	M	M	H		N/A		2	9.2	18	84.0
35	E	15	M	M	A	65	153.5	249	0	4.7	14	77.3
36	E	5	M	M	N		N/A		0	1.1	2	97.7
37	E	5	M	M	H		N/A		0	1.1	3	97.0
38	E	5	M	M	A	64	137.6	237	0	0.5	2	96.3
39	E	10	R	M	N		N/A		1	4.1	8	78.0
40	E	5	N	N	N		N/A		0	1.9	4	99.7
41	E	10	N	N	N		N/A		1	7.3	14	99.3
42	E	15	N	N	N		N/A		3	16.4	34	99.3
43	E	20	N	N	N		N/A		5	29.3	56	99.0
44	E	10	R	M	H		N/A		1	4.1	9	88.0
45	E	10	R	M	A	66	152.1	241	0	2.1	6	85.3
46	E	15	R	M	N		N/A		2	9.3	20	63.7
47	E	15	R	M	H		N/A		2	9.2	18	83.7
48	E	15	R	M	A	65	154.0	249	0	4.8	14	77.0
49	E	5	R	M	N		N/A		0	1.1	3	98.0
50	E	1	M	M	N		N/A		0	0.1	1	99.7

Table A.4. Summary of Experiments: Test 51 to 78.

Test #	Dataset	Candidates	Distance Function	Tau (τ) Function	Classifier	A^* Depth (nodes) MIN	A^* Depth (nodes) AVG	A^* Depth (nodes) MAX	Runtime (seconds) MIN	Runtime (seconds) AVG	Runtime (seconds) MAX	% Correct
51	E	1	M	M	H		N/A		0	0.1	1	99.7
52	E	1	M	M	A	75	136.8	226	0	0.1	1	99.3
53	E	1	R	M	N		N/A		0	0.1	1	99.7
54	E	1	R	M	H		N/A		0	0.1	1	99.7
55	E	1	R	M	A	76	137.1	227	0	0.0	1	99.3
56	E	1	N	N	N		N/A		0	0.2	1	100.0
57	E	1	N	N	H		N/A		0	0.1	1	99.3
58	E	1	N	N	A	91	134.6	227	0	0.1	1	99.7
59	E	1	N	T	N		N/A		43	215.6	736	100.0
60	E	1	N	T	H		N/A		45	214.1	720	99.3
61	E	1	N	T	A	65	90.9	168	7	63.6	298	99.7
65	E	5	N	Z	N		N/A		0	1.0	2	99.7
66	E	10	N	Z	N		N/A		0	3.4	8	99.3
67	E	15	N	Z	N		N/A		1	7.5	15	99.3
68	E	20	N	Z	N		N/A		3	13.2	29	99.0
69	E	5	N	Z	H		N/A		0	1.0	2	98.7
70	E	10	N	Z	H		N/A		0	3.5	7	98.7
71	E	15	N	Z	H		N/A		2	7.4	15	98.7
72	E	20	N	Z	H		N/A		2	13.2	27	98.7
73	E	5	N	Z	A	92	139.4	235	0	0.5	1	99.0
74	E	10	N	Z	A	93	145.0	244	0	1.8	4	99.0
75	E	15	N	Z	A	94	147.2	246	0	4.0	8	98.7
76	E	20	N	Z	A	96	148.4	250	1	7.0	15	98.7
77	E	5	R	M	H		N/A		0	1.1	3	97.0
78	E	5	R	M	A	65	138.0	238	0	0.5	2	96.3

Table A.5. Summary of Experiments: Test 80 to 108.

Test #	Dataset	Candidates	Distance Function	Tau (τ) Function	Classifier	A^* Depth (nodes)			Runtime (seconds)			% Correct
						MIN	AVG	MAX	MIN	AVG	MAX	
80	E	20	N	T	A	76	101.9	181	3059	25856.1	108618	99.0
81	E	15	N	T	A	76	101.4	180	1711	14425.7	61283	99.0
82	E	20	N	T	H		N/A		16232	78402.3	233270	98.7
83	E	15	N	T	H		N/A		8877	44075.5	131601	98.7
84	E	20	N	T	N		N/A		16226	78112.0	233602	99.7
85	E	15	N	T	N		N/A		9238	43871.0	131959	99.7
90	H	1	N	T	N		N/A		44	519.4	4672	53.6
91	H	1	N	T	H		N/A		44	519.4	4688	52.9
92	H	1	N	T	A	72	155.3	233	10	269.4	3319	55.7
93	H	1	N	N	H		N/A		0	0.1	1	52.9
94	H	1	N	N	A	102	211.9	281	0	0.1	1	55.7
95	H	1	N	N	N		N/A		0	0.1	1	53.6
96	H	5	N	T	N		N/A		1114	8437.8	54916	50.0
97	H	5	N	T	H		N/A		1102	8470.1	56848	54.3
98	H	5	N	T	A	78	156.8	241	239	4403.6	33299	53.6
99	H	5	N	N	H		N/A		0	1.7	3	57.1
100	H	5	N	N	A	114	209.8	286	0	1.2	3	56.4
101	H	5	N	N	N		N/A		0	1.7	3	56.4
102	E	1	N	Z	N		N/A		0	0.1	1	100.0
103	E	1	N	Z	H		N/A		0	0.1	1	99.3
104	E	1	N	Z	A	87	131.1	225	0	0.0	1	99.7
105	E	1	M	Z	N		N/A		0	0.1	1	99.7
106	E	1	M	Z	H		N/A		0	0.1	1	99.7
107	E	1	M	Z	A	75	136.8	226	0	0.1	1	99.3
108	E	5	M	Z	N		N/A		0	1.1	3	97.7

Table A.6. Summary of Experiments: Test 109 to 133.

Test #	**Dataset**	**Candidates**	**Distance Function**	**Tau (τ) Function**	**Classifier**	**A^* Depth (nodes)**			**Runtime (seconds)**			**% Correct**
						MIN	AVG	MAX	MIN	AVG	MAX	
109	E	5	M	Z	H		N/A		0	1.0	2	97.0
110	E	5	M	Z	A	64	137.5	237	0	0.5	2	96.3
111	E	10	M	Z	N		N/A		1	3.8	8	78.0
112	E	10	M	Z	H		N/A		1	3.9	9	88.7
113	E	10	M	Z	A	66	151.6	240	0	2.0	6	85.7
114	E	15	M	Z	N		N/A		2	8.5	17	64.3
115	E	15	M	Z	H		N/A		2	8.6	16	84.0
116	E	15	M	Z	A	65	153.4	249	0	4.4	13	77.3
117	E	20	M	Z	N		N/A		4	15.0	29	59.7
118	E	20	M	Z	H		N/A		3	15.0	28	83.0
119	E	20	M	Z	A	65	153.8	250	0	7.8	24	76.0
120	H	20	N	N	N		N/A		7	25.4	46	37.9
121	H	20	N	N	A	113	211.9	276	2	17.6	40	42.9
122	E	1	R	Z	N		N/A		0	0.1	1	99.7
123	E	1	R	Z	H		N/A		0	0.1	1	99.7
124	E	1	R	Z	A	76	137.0	227	0	0.0	1	99.3
125	E	5	R	Z	N		N/A		0	1.1	2	98.0
126	E	5	R	Z	H		N/A		0	1.1	2	97.0
127	E	5	R	Z	A	65	137.9	238	0	0.5	2	96.3
128	E	10	R	Z	N		N/A		1	4.0	8	78.0
129	E	10	R	Z	H		N/A		1	3.9	8	88.0
130	E	10	R	Z	A	66	152.1	240	0	2.0	6	85.7
131	E	15	R	Z	N		N/A		1	8.5	16	63.7
132	E	15	R	Z	H		N/A		1	8.5	16	83.7
133	E	15	R	Z	A	65	153.9	249	0	4.4	13	77.0

Table A.7. Summary of Experiments: Test 134 to 156.

Test #	**Dataset**	**Candidates**	**Distance Function**	**Tau (τ) Function**	**Classifier**	**A^* Depth (nodes)**			**Runtime (seconds)**			**% Correct**
						MIN	**AVG**	**MAX**	**MIN**	**AVG**	**MAX**	
134	E	20	R	Z	N		N/A		3	15.0	31	58.3
135	E	20	R	Z	H		N/A		3	15.2	31	82.7
136	E	20	R	Z	A	65	154.3	250	0	7.7	22	75.3
137	H	1	N	Z	H		N/A		0	0.1	1	53.6
138	H	1	N	Z	A	97	210.4	280	0	0.1	1	55.7
139	H	1	N	Z	N		N/A		0	0.1	1	53.6
140	H	5	N	Z	H		N/A		0	1.0	2	57.1
141	H	5	N	Z	A	109	207.6	286	0	0.7	2	56.4
142	H	5	N	Z	N		N/A		0	0.9	2	56.4
143	H	10	N	Z	H		N/A		1	3.3	6	50.7
144	H	10	N	Z	A	103	208.8	282	0	2.2	5	52.1
145	H	10	N	Z	N		N/A		1	3.3	6	47.1
146	H	20	N	Z	N		N/A		3	12.2	21	37.9
147	H	20	N	Z	H		N/A		4	12.6	23	42.1
148	H	20	N	Z	A	107	209.8	276	1	8.7	19	43.6
149	H	1	M	M	H		N/A		0	0.1	1	76.4
151	H	5	M	M	H		N/A		0	1.0	2	73.6
152	H	10	M	M	H		N/A		1	3.6	7	55.7
153	H	15	M	M	H		N/A		2	8.0	14	42.9
154	H	1	R	M	H		N/A		0	0.1	1	76.4
155	H	5	R	M	H		N/A		0	1.0	2	72.1
156	H	10	R	M	H		N/A		1	3.5	7	54.3

REFERENCES

[1] D. C. Tech. (2013) Rip original palm os. [Online]. Available: http://www.techdc.com/rip-original-palm-os

[2] V. I. Pavlovic, R. Sharma, and T. S. Huang, "Visual interpretation of hand gestures for human-computer interaction: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 677–695, Jul 1997.

[3] J. Alon, V. Athitsos, Q. Yuan, and S. Sclaroff, "Simultaneous localization and recognition of dynamic hand gestures," *IEEE Motion Workshop*, pp. 254–260, Jan 2005.

[4] ——, "A unified framework for gesture recognition and spatiotemporal gesture recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 9, pp. 1685–1699, 2009.

[5] F.-S. Chen, C.-M. Fu, and C.-L. Huang, "Hand gesture recognition using a real-time tracking method and hidden markov models," *Image and Video Computing*, vol. 21, no. 8, pp. 745–758, Aug 2003.

[6] M. J. Black and A. D. Jepson, "Recognizing temporal trajectories using the condensation algorithm," *Automatic Face and Gesture Recognition*, pp. 16–21, 1998.

[7] Q. Yuan, S. Sclaroff, and V. Athitsos, "Automatic 2d hand tracking in video sequences," *Proceedings of the Seventh IEEE Workshop on Applications of Computer Vision*, 2005.

[8] P. Senin, "Dynamic time warping algorithm review," 2008.

[9] J. B. Kruskall and M. Liberman, *The symmetric time warping algorithm: From continuous to discrete.* Addison-Wesley, 1983.

[10] L. Rabiner and B. Juang, *Fundamentals of Speech Recognition.* Prentice Hall, 1993.

[11] T. Darrell and A. Pentland, "Space-time gestures," *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 335–340, 1993.

[12] A. Corradini, "Dynamic time warping for off-line recognition of a small gesture vocabulary," *Proceedings IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, pp. 82–89, 2001.

[13] J. Yao, C. Lin, X. Xie, A. J. Wang, and C.-C. Hung, "Path planning for virtual human motion using improved a^* algorithm," *Seventh International Conference on Information Technology*, 2010.

[14] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach.* Upper Saddle River, New Jersey, 07458: Prentice Hall, 2003.

[15] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Upper Saddle River, New Jersey, 07458: Prentice Hall, 2003.

[16] L. G. Shapiro and G. C. Stockman, *Computer Vision.* Upper Saddle River, New Jersey, 07458: Prentice Hall, 2001.

[17] J. R. Hubbard and A. Huray, *Data Structures with Java.* Upper Saddle River, New Jersey, 07458: Pearson Education Inc., 2004.

[18] Itseez. (2013) Opencv: Open source computer vision. [Online]. Available: http://opencv.org

[19] B. Dawes, D. Abrahams, and R. Rivera. (1998-2007) Boost c++ libraries. [Online]. Available: http://www.boost.org

[20] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001.

[21] G. Farneback, *Two-frame motion estimation based on polynomial expansion*, ser. Lecture Notes in Computer Science, 2003, no. 2749.

[22] A. A. Efros, A. C. Berg, G. Mori, and J. Malik, "Recognizing action at a distance," *Proceedings of the Ninth IEEE International Conference on Computer Vision*, pp. 726–733, 2003.

BIOGRAPHICAL STATEMENT

After a 13 year career in Information Technology, Danny Allen Hanson received his Bachelor's degree in Computer Science from California State Polytechnic University, Pomona in 2006. Danny's undergraduate research was in robotics with an emphasis on path planning and he is currently working in robotics and computer vision. He received his Masters of Science in Computer Science with a focus on computer vision at the University of Texas at Arlington in May 2013. His work focuses on developing cutting-edge technologies for military applications.