Behavior Model for Predictive Tracking Of Multiple Objects

Author: James Logan
UCCS Computer Science Masters Candidate
University of Colorado, Colorado Springs
Colorado Springs, Colorado United States
jlogan@uccs.edu

Advisor: Dr. Terrance Boult
UCCS El Pomar Professor
University of Colorado, Colorado Springs
Colorado Springs, Colorado United States
tboult@vast.uccs.edu

Committee Member: Mr. Al Brouillette
University of Colorado, Colorado Springs
Colorado Springs, Colorado United States
abrouill@uccs.edu

Committee Member: Dr. Edward Chow
University of Colorado, Colorado Springs
Colorado Springs, Colorado United States
ccchow@uccs.edu

Abstract—This document contains information on implementing a multiple camera multiple human tracking system prototype. This document is being submitted as a Masters’ Project report to be reviewed as a qualification for the author’s candidacy for a Masters in Computer Science. The reviewer of this document should have a basic background in areas of Computer Vision, networking protocols, C++, and visual tracking techniques.

I. INTRODUCTION

The concept of this project stems from Capture Agents associated with the Matterhorn project. Capture Agents are hardware and software units employed by universities and small institutions to aid scheduled autonomous lecturer recordings[3]. Two basic configurations for this type of system exist. The first being a statically mounted camera (the director) that has a wide angle view of the environment controlling a Pan Tilt Zoom (PTZ) camera (the cameraman). The second utilizing a single PTZ camera focused on the general direction of the speaker. Both configurations are designed to assume that there is only one object in the environment making their tracking capabilities limited and unpredictable should multiple lectures become present in the scene. The tracking and modeling methods implemented by these Capture Agents are not robust enough to discriminate between its current track and other similar or foreign tracks that it may encounter[1][2][4][5][6].

The emphasis of this project is to evaluate the current tracking methods used by the Capture Agents to aid in developing a prototype system with improved detection and tracking methods. In addition, its design will encompass multiple PTZ cameras to be attached to the system where each will act as its own directing and cameraman unit eliminating the need for secondary camera support. The prototype system will introduce object discrimination and filtration mechanisms to allow cameras to be focused on a unique object and aid in preventing other PTZ cameras from tracking the same subject. Accomplishing this task will require a survey of recognized Computer Vision techniques for modeling, tracking, and classification and assessing whether or not a notable approach is appropriate for this task. The objective of this project is to be an educational reference in visual based tracking. It demonstrates through combining and taking advantage of known Computer Vision techniques it is possible to create an effective multi-object tracking system.

The following sections in this paper describe the research, integration, and implementation of the visual tracking system prototype along with possibilities of future work and the issues observed while developing the prototype system.

II. COMPONENT INTEGRATION

Component integration began with installing OpenCV with its supporting libraries (See Appendix A for install order). Versions 2.4.5, 2.4.3, 2.4.2, 2.3.0, and 2.2.0 were installed and tested on Red Hat and Debian based Linux systems. It should be noted that APT, YUM, YaST were not used for installation of OpenCV or its supporting libraries. Instead a manual install was used for the software libraries to allow the compilation settings to be tuned for the current system. With Red Hat based distributions OpenSUSE, Redhat 18 and CentOS 6.4, OpenCV had issues with installing version 2.4.5 and executing a test program with a network camera on all previously mentioned versions. Version
2.4.5 had issues with Python library locations when running CMAKE. Python environment variables PYTHON_EXECUTABLE, PYTHON_INCLUDE, PYTHON_LIBRARY, PYTHON_PACKAGES_PATH, and PYTHON_NUMPY_INCLUDE_DIR where altered for CMAKE which lead to the version being installed successfully. All versions of OpenCV on the Red Hat distributions exhibited the same behavior when running a test program consisting of an image query and running the results through a detection method. The results of the program would either be a null image, a black image, or the system would fault and exit. Using OS commands like netstat and the network tool Wireshark confirmed a connection to the network camera however the data was not being processed correctly.

After reviewing OpenCV blogs, there was constant mention of utilizing Ubuntu 11.04 with network cameras. Therefore it was tested. OpenCV versions 2.3.0 and 2.4.5 would not install successfully even with the modifications to the Python CMAKE variables. These versions would get just above 90% of the install fail and exit immediately. No logs could be found for the errors that occurred. OpenCV 2.4.3 installed successfully however when the test program was executed the program terminated with error: 0xC000001D Illegal Instruction. OpenCV documentation confirmed that is a known issue: Bug #2591- running cvHaarDetectObjects or DetectMultiscale. This was caused by the detection method used in the test program. Version 2.4.2 had all the same issues as observed on the Red Had distributions. OpenCV 2.2.0 however, installed and executed the test program successfully. There was key difference noted when installing OpenCV on Ubuntu vs. the Red Hat Distributions. The Video For Linux (V4L/V4L2) driver modules were not installed on Ubuntu. A test whether or not the V4L/V4L2 driver modules have any impact on OpenCV for the mentioned Red Hat distributions will be conducted at a later time.

Initial testing of the cameras showed that test program could process H.264 and MPEG4 over RTSP and MJPEG over HTTP at QVGA resolution. However due to hardware limitations the framerate, image quality, and video preference (motion smoothness vs. image quality) had to be adjusted to minimize network lag and pixilated images. The framerate had to be set to 15fps or less, image quality 256Kbps or less, and video preference was set to medium or lower. If the values of those settings were exceeded a 20 second or greater network lag would occur and the program console output would report corrupted macro-blocks resulting in the pixilation of the output image. The data corruption would eventually get to a point that the detection method in the test program would no longer work. Objects in the output image were so pixelated that they were no longer identifiable.

Once the video settings for the camera were defined the next step was to figure out how to control the cameras. Both cameras are ZyXEL IPC4605N PTZ network cameras and run a mini-http server. The mini-http server uses basic http authentication and accepts query strings through CGI to control its movements. This was discovered by monitoring the communications between the client machine and mini-http server via Wireshark. The packets captured showed in plain text the query strings necessary to control the cameras’ movements. Implementing how to control the cameras’ motion required use of the cURL library. cURL is a client side library that supports many internet protocols and was specifically selected because it supports basic http authentication that allows a query string to be sent with the basic authentication request. There were no issues integrating the library into the project source code. The only issue was observed with cURL was during program execution. If the program attempted connect to a non-existent IP or the query string was malformed, cURL would cause the program to terminate prematurely.

III. CURRENT SYSTEM METHODOLOGIES

The Capture Agent systems are geared to operate in environments containing static objects excluding the speaker. They implement either a facial recognition or motion based algorithm to track objects. Both implementations lack tracking and modeling capabilities and are prone to unpredictable behavior when multiple speakers are in the field of view. Another disadvantage is the requirement of using two cameras to track one lecturer. The statically mounted camera assumes the responsibility of guiding the PTZ camera while maintaining state information of the current scene. This process running concurrently with the PTZ camera feed generates unnecessary overhead processing for the Capture Agent. The system will have to evaluate two camera streams and synchronize the data to ensure the camera is tracking properly. In contrast, the single PTZ configuration has no guidance when observing the scene and has the potential to lose tracking capabilities permanently if it becomes disoriented.

The basic facial recognition version of the Capture Agent, although not documented in the specifications comes across as a Viola-Jones based technique. The basic facial recognition aspect of the design is susceptible to occlusion, illumination variance, orientation changes, expressions (gestures), skin tone, blurriness, and camera zooms. Another drawback to this approach is the number of key points or features detected. Limiting the detection to the face does not
allow the system to maintain an accurate model of the subject. The tracking capabilities in this approach are mainly affected by changes in the objects’ orientation and multiple speakers in the field of view. If a speaker turns and faces away from the camera and moves around, the system would lose its current subject or randomly select another face to track.

The alternate method for Capture Agents is a Motion Histogram Based Detection scheme. The algorithm is simple and has more tolerance to illumination changes than the facial detection. The concept is motion pixels that represent a percentage of a region are considered to be the speaker. This idea is suited for statically mounted cameras. It requires a static background to determine movements and is similar to frame differencing or optical flow and is predisposed to false positive detection. The major concern in regards to tracking comes from potential noise in the image. If any type of motion is observed, the system may consider that region of pixels the speaker and adjust the PTZ camera accordingly. This method lacks the ability to maintain any type of state information on its object track.

IV. ALGORITHM REVIEWS

Features (encoding the visual appearance based on identification), Classifiers (determining whether or not an object is a person), Modeling (maintaining information generated by the classifying algorithm), and Model Tracking (acquiring coordinate information and predicting the placement of the model on the current image) were essential in building a successful visual tracking prototype. A survey on computer vision algorithms was conducted to determine the best approach prior to implementation. For human detection algorithms involving frame differencing, background subtraction, Histogram or Oriented Gradients (HOG), Partial Least Squares, HAAR Wavelets with SVM, Scale Invariant Feature Transform (SIFT), Speeded Up Robust Features (SURF), Hough Transforms, and Active Contour Modeling. As for human tracking, Kalman Filtering, Mean Shift, Optical Flow, Contour Tracking and Human Matching were reviewed. Notable methods reviewed for tracking and detection included:

- Beleznai 2004: Focused on finding isolated human in an environment. Quick Analysis: More suited for the old system but would not work well with a multiple human environment unless humans were distinctly spaced[22].
- Haga 2004: A three part process that relies on temporal and spatial human motion characteristics using background subtraction and classification. Quick Analysis: The disadvantage here is when the object stops moving it can become part of the background model. There is also a fair amount of overhead processing due to background subtraction[23].
- Eng 2004: Region based background modeling method that compares updated images to a current background model and classifies head or body features based on change in pixel information. Quick Analysis: Suited for a statically positioned camera. Implementing this method with a PTZ camera would require the current background model to be invalidated and a new one generated once a pan, tilt, or zoom operation was executed. This involves unneeded overhead processing[24].
- Elzein 2004: Uses optical flow with frame differencing. Requires a point of reference to determine an objects velocity. Groups blobs by morphological operations followed by classification. This requires a statically mounted camera. Quick Analysis: There are two major problems with this approach. The first being if this was PTZ implementation how would a reference point be chosen? The second being any foreign object that could change the way the blob grouping is shaped could cause a false negative[25].
- Toth and Aach 2003: A background subtraction method that uses frame differencing. Quick Analysis: The concept is to look for absolute differences between frames and classify blobs using connected components via a neural network. A neural network is impractical and foreign objects introduce blob misclassifications[26].
- Lee 2004: Shaped based approach using background subtraction via frame differencing. The results are classified by their contour boundary. Quick Analysis: This method requires overhead preprocessing and has the same issues as blob grouping in terms of a misclassification of foreign objects changing the shape of the contour boundary[36].
- Zhou and Hoang: Uses background subtraction and temporal differencing with a Codebook approach to determine human objects. In addition it uses tracking using color histograms of the motion blobs. Quick Analysis: Foreign objects can cause false negatives in classification[27]. The color based tracking method prone to illumination variances and occlusion.
- Yoon and Kim 2004: Simple concepts of skin color and motion information. Quick Analysis: This idea is too simplistic. Issues range to minimal moving object producing little to no motion information to illumination and occlusion issues[28].
- Xu and Fujimura 2003: Background elimination techniques followed by ellipse fitting to silhouettes and classification performed by a heuristic algorithm focused on movement. Quick Analysis: Overhead preprocessing and a best guess
approaches are taken to determine human movement[29].

- **Cutler and Davis 2000**: Based on biological movements. Uses frame differencing to determine image region movements. A lattice fitting structure for classification is utilized. **Quick Analysis**: Overhead processing in terms of what constitutes a biological movement and segmentation of image regions[39].

- **Utsumi and Tetsutani 2002**: Technique involves calculating geometric distances of body parts (distance mapping) and a fairly large database of human and non-human distances to generate a statistical model of the current image. **Quick Analysis**: Impractical having to constantly access a database to construct a potential model[30].

- **Viola 2003**: Detection of humans in static images. Uses cascade classifiers to determine human. **Quick Analysis**: Limited on the number of features it obtains since it is primarily used for facial recognition and detection on static images [37].

- **Dalal and Triggs 2005**: Histogram of Oriented Gradients. This algorithm was the one chosen and will be discussed later in the paper[12].

- **Sidenbladh 2004**: Uses a SVM on trained optical flows to classify human objects. **Quick Analysis**: Once an object has little or no motion it becomes part of the background[31].

A majority of the techniques mentioned above can be implemented with OpenCVs’ standard libraries using a combination of different methods (Kalman, CAMShift, Frame Differencing, Codebooks (Bag of Words), Lucas-Kanade, contour and blob operations). The greater part of these types of tracking techniques emphasize on user defined parameters (color, reference points, basic shapes) for an environment and/or adaptive background modeling. It was observed that tracking schemes that performed classification had one or two intermediate steps before the object was classified. The proceeding steps normally consisted of detecting motion to isolate the object from the background then either segmentation or grouping of pixels for analysis prior to classification. Intermediate steps that fail, causes the process as a whole to fail and introduce overhead processing. For the prototype system in this project these types of designs were not a desirable approach. The incentive was to have a scheme that classified then tracked based on classification resulting in HOG to be chosen. The HOG function requires a single image to perform analysis on and does not need any pre or post processing to perform classification. It has adjustable parameters for the search window, scaling, grouping (combining features), and SVM (object detection method). In addition it provides the image coordinate information about features it has classified. This is a great benefit when tracking a desired object. Paired with a mechanism that can maintain statistical information and predict an objects position within the image, the HOG method provides an advantageous approach with tracking efficiency. As a side note, there is current research that involves SIFT and SURF implemented with a SVMs. Currently known SIFT and SURF applications extract features from the entire image and perform no classification on said features. Thus, some of the concerns that arise are training sets used for the SVM (MIT or INRIA), and how features are grouped and processed prior to classification[21]. These feature extraction methods were considered in the development of the prototype system but were waived due to the aforementioned concerns.

V. **THE PROTOTYPE SYSTEM**

The prototype system takes advantage of a HOG mechanism that enables the system to identify and classify features and extract coordinate and model feature information. It is designed to eliminate the need for reference images, adaptive background modeling, and cascading image processing. The prototype promotes a generic configuration setup to allow a variety of camera manufactures to connect assuming they can support the configuration parameters. In addition, The GNU Scientific Libraries (GSL) are integrated into the system to provide histogram statistics for tracking points and functions for predicting object positions within an image. The prototype system introduces a dual use of an open source Fast Match Template function designed and slightly modified to aid in model filtering and discrimination. Overall the prototype consists of three major components:

- **Camera Controller** – provides all the camera connection, control, image querying information and defines all global configuration rules for the system (Reference Appendix D for Camera Controller flow diagram).

- **HOG Processor** – Generates coordinate and timestamp information for the feature data classified by the HOG SVM (Reference Appendix D for HOG Processor flow diagram).

- **Track Analyzer** – Behavioral component of the system. It administers all actions to be executed by the camera based on image data and the rules defined by the Camera Controller (Reference Appendix D for Track Analyzer flow diagram).

A. **Prototype System Execution Overview**

The prototype system begins execution using Xerces 3.1.1 (Open source DOM library for parsing xml files) to set the parameters defined in the
cam_system.xml (prototype system configuration file) to set up a Camera Controller object for each camera attached to the system. A list of the system parameters can be found in Appendix B of this document. Following the initial setup each camera will then be assigned a HOG Processor and Track Analyzer component. The HOG Processors’ detector method (people, animal, car, etc) will then be set. Finally each Track Analyzer will be assigned a Camera Controller object and a global vector object containing data structures designed to allow multiple Track Analyzers to share model information.

The basic flow of execution is to query an image run the image through the HOG Processor and finally allow the Track Analyzer make decisions based on the HOG results on a per frame basis. The analysis on each frame is to allow each camera to maintain real time accurate representation of the model that it is tracking. Each camera will be in either an active or inactive tracking state while the system is processing data.

1) Inactive State Processing: In an inactive state image processing begins once the image is queried and passed to the HOG Processor. The HOG Processor will then run the detector method on the image and bound any resultant features with rectangles and give each feature a timestamp. The processor will place data from the feature information collected into data structures that is then passed to the Track Analyzer for evaluation.

Before the Track Analyzer assesses that HOG Processor data it must be pre-processed. During integration and testing it was discovered that the HOG function can potentially have two anomalous outputs. The first is the multiple rectangular groupings of a single object. The HOG function has a grouping parameter that controls this property. However, it does not always, and must be compensated for. Rectangle clustering was implemented to solve this issue. All center points of the rectangles within a defined radius of the first rectangle where the features were classified are clustered. The second HOG anomaly has to do with the HOG functions’ inability to keep bounding information within the dimensions of the image. This event usually occurs when an object enters or exits the field of view resulting in bounding rectangles to exceed the image boundary. In this case it was necessary to resize the bounding rectangles to images’ edges. This is a critical check since the Track Analyzer needs to access image matrix data based on where the bounding information occurs otherwise a memory access violation would occur. After clustering and edge checking are performed, the system determines if the camera has an active track and regions of interests (ROIs) are generated based on the HOG Processors’ results. If no active track is present, the Track Analyzer will review and filter known models in the shared vector object. The ROIs generated are compared to the current tracked models and if any ROI matches a defined percentage of a tracked model, it becomes disqualified potential tracking. If any ROI models remain, one will be selected at random. Once a model is selected the Track Analyzer will submit the ROI model to the shared vector with cameras information, place the camera in an active tracking state, initiate time monitoring, acquire coordinate information, and adjust the camera to have the object in the confinement zone (user defined region of the image where the camera wants the target object to be located) if necessary. If no track is available the Track Analyzer will continue to process HOG Processor results on each frame until a track is available.

2) Active State Processing: A Track Analyzer with a camera in an active tracking utilizes the same preprocessing clustering and resizing as a camera in an inactive state. Once the HOG data is processed, the track analyzer will use the GSL functions on x and y position histograms to obtain a predicted position. A subset of models in the predicted image region will be subject to discrimination against the current tracked model. Using the Fast Match Template function the track analyzer can determine based on the highest percentage which particular ROI model is likely to be a representation of the current model. The model is updated with the ROI model and the Track Analyzer then determines if a camera operation is necessary (pan, tilt, zoom). In the event that the Track Analyzer is unable to confirm its track within a given time period it will purge all model and position data and reset the camera to its home position and put in an inactive tracking state. Zooming was not integrated into the prototype system since the cameras had their own auto zoom capabilities. However, there is a parameter for zooming in the system configuration file to support other cameras. The software can be readjusted if necessary for this capability by code changes. A possible adaptation would consist of zooming out being dependent upon the number of pans or tilts within a given time frame while zooming in could rely on current discrimination values weighed against a defined threshold value.

3) Camera Control: Controlling the cameras’ movements are based on the state of the camera (active vs. inactive). When a camera transitions to an active state it immediately tries to assess if a pan or tilt operation is necessary to put the subject within the boundaries of the confinement zone given the initial coordinates. A Track Analyzer with a camera in an already active state takes timing, object position, and frames processed since last camera movement into account before making its decision to adjust the camera. Conditions to pan or tilt are defined when a subject is outside the area of the confinement zone for a specified period of time. A drop track state occurs when the HOG processor does not produce sufficient data for the Track
Analyzer to obtain the model data needed to maintain its current track. When a camera motion is executed all data from the x and y position histograms are purged and the search area restriction is lifted to allow the new coordinate information to be obtained. Once acquired the search restriction is placed back on the tracking camera. The procedure is necessary since the system does not maintain any type of reference point. Timing and counting subsequent frames after a camera movement is vital in preventing the Track Analyzer from over correcting the camera. Timing also plays a role in restraining the system from issuing an undesired drop track command. During integration and testing an evident behavior was observed regarding the HOGs’ SVM. Even though all frames passed through the SVM, there was no guarantee that any model data per each frame processed would be returned. This resulted in placing a defined timeout condition in the system to give the SVM time to generate data.

B. Fast Match Template Function Overview

The Fast Match Template function is an open source method augmenting OpenCVs’ match template function. Its original use was in aerial ground searches to determine if a defined feature set exists over a region and returns a percentage based on the confidence of the compared data. It was adapted for the prototype system to aid in filtration and discrimination of model data. The concept of the Fast Match Template is to search for a target image inside a source image and return true or false if a specified percentage of the target is found in the source. This approach works for filtration, however it had to be altered for discrimination. An additional parameter was added to the function signature that allowed the actual percentage of the target and source comparison to be captured. This value allows the Track Analyzer to select the best representation of the subset of ROI models it receives when updating the current track model and ensure the tracks’ quality. Also, the Fast Match Template utilizes down sampling using a pyramid scheme. The down sampling can be specified in the function parameters. The benefits provided by this operation in terms of down sampling are reductions in noise and search area. The only requirement necessary for this function are the dimensions of the source and target image. The target image has to be equal or lessor than the source image to perform the necessary operations. As a pre-caution the Track Analyzer resizes all ROI model images prior to using Fast Match Template. In essence it is utilized to ensure the quality of the cameras track.

VI. TESTING AND OBSERVATIONS

Evaluating the prototype system consisted of three visual tests.

A. Visual Object Detection and Processing

The first test was an assessment using one human subject to confirm the systems’ ability to:
- confirm the SVM of the HOG Processor was able to extract human model features from a frame
- demonstrate the shared model information was distributed so that all cameras excluding the tracking camera attached to the system would not track the single human subject
- ensure the cameras’ Track Analyzer component could update the tracked model in real time
- discriminate the real time model information provided by the HOG Processor with the aid of the integrated Fast Match Template function against the current tracked model.

Results obtained demonstrated human subject detection and model generation that could be updated in real time. It also exhibited the ability to show off the filtration and shared data mechanisms that prevent multiple cameras from tracking a single object. Approximately 900 discrimination values were collected when conducting the test and a noticeable pattern was recognized. The percentage of each discrimination value was based on the orientation of the subject. Percentages of 70 to 100 were found when a subject had steady orientation while 40 to 69 were output when a change in orientation occurred and anything below 40 percent usually was a result of entering or exiting the scene. Note as a subject’s orientation stabilized the percentage would rise. Tests two and three were extensions of test one to confirm the systems’ overall functionality.

B. Multiple Camera and Subject Tracking

The second test was a full system test using multiple human subjects, all cameras, and determining if filtration would execute correctly. Finally, a third test consisting of the systems’ capability to keep a tracked subject within the area of the confinement zone was conducted. This was a minor test involving moving in and out of the confinement zone and testing the systems’ reaction based on the user defined parameters for the movements.

Conducting tests two and three resulted in a runtime bug with tracking and model misinterpretations being discovered. The tracking bug found in test three occurred when the system initially begins to track a subject. As data points are collected and placed into the GSL bins for the histograms, there is the possibility that the Track Analyzer will incorrectly predict the anticipated position of the object. GSL documentation states that certain operations performed on bins with
equal weights will return the first occurrence of a particular weighted bin at the specified weight. This characteristic was apparent when an object would enter a scene and continue to move across the frame. The system would have bins with equal weights in the x direction and the predicted versus actual position was completely off. To compensate, the system compares the last known position obtained from the HOG processor results and the predicted location. If the difference is greater than the defined variance then the last known values for the subject are used instead.

Dual tracking test two created circumstance that made the system assume that it was analyzing two different environments. A PTZ with a light sensor triggered the camera to automatically adjust for a low level setting. The event caused a significant reduction in feature information that caused multiple cameras to track one target and created difficulties for the camera to maintain its current track due to subjects having a high degree of similar pixel information. This outcome was unexpected and could not be resolved since there was no method to control the auto lighting feature on the camera.

C. Common System Behaviors

Each test was conducted several times to assess the behavior of the system. In general, each test passed based on their respective criteria and noted bug fixes. However there were factors observed that influenced how the system detected and tracked objects. These factors were not isolated to single test but were observed on all tests and their cause was based on either hardware or software components of the system.

1) HOG SVM Issues: The first notable incident was a system miscue off a human silhouette. Back lighting in the environment caused a projection a human shadow on to a surface scene which triggered the system to track. Although the system is working correctly, it however created a misrepresentation or false feature set of a model. The second issue was a false positive detection by the HOG Processor. HOG has an accuracy rate approximately 85 to 90\%[38] percent using recommended parameters, therefore a misclassification was expected to happen during testing. The inanimate objects detected by the system usually had human like dimensions.

2) PTZ Issues: Lastly, a majority of the atypical behavior witnessed was due to delayed and corrupted frames from the network cameras. Delayed frames were apparent in two ways. The first, being a constant delay between frames and the asynchronous data flow from the network cameras. The system polls each camera equally however the trend observed at QVGA and higher resolutions was that one camera was always ahead of the other when displaying information.

This behavior depending on the degree of delay could cause a dual tracking status of one object because the filtration processes did not have enough data to filter out the object. Handling this issue resulted in a code augmentation for model crosschecking. Periodically the system will require the camera to check their model pixel information to other models being tracked. Cameras that meet the criteria of a possible dual track will drop their model and reset. The second, in contrast was sporadic delay. A frame would be received by the system followed by random delay and concluding with several frames being sent from the camera. If an object was in track and the Track Analyzer had to make an adjustment to keep it in the confinement zone, it would over adjust and either turn away from the environment or lose the track completely even though the subject remained in the environment. In both cases a drop track condition would be likely executed by the Track Analyzer. Corrupted frames on the other hand had a unique effect on the system when using multiple cameras. If one camera was in track and had a decent uncorrupted model while another camera was sending corrupted frames to the system there was a potential that the filtration process could be bypassed resulting in two or more cameras tracking one object. Although a frame was corrupted there was still enough information for the HOG processor to detect a human subject but it was altered enough by the corruption that it did not meet the filtration requirements for exclusion. When investigating the corruption, it was discovered that one of the PTZ cameras was failing. Observing the network stream showed the healthy camera dominating the stream with a constant packet size while the declining camera would send packet sizes less than half of what the healthy one would produce. Eventually the failing camera would stop generating packets all together would have to be replaced. The ZyXELes were replaced with two D-Link 5020L network cameras to continue and complete system testing. An intriguing note on corrupted frames is FFmpeg ability to detect the problem however when passed to OpenCV the information is seen as is. In the future there could be work done to correct the disconnection between OpenCV and its supporting FFmpeg library. Given the degree of corruption, FFmpeg should pass along information in regards to the queried frames’ status that allows OpenCV to determine if it needs to be discarded. As a pre-caution, code was inserted to ensure a complete frame was acquired prior processing to mitigate partial frame processing.

VII. PERFORMANCE EVALUATION

Data was collected to assess the performance different components of the prototype system including the test hardware platform itself. Component analysis focused behaviors of the HOG function, the Track
Analyzer’s post processing of HOG Processor data and object coordinate tracking and prediction, and runtime I/O of the hardware test platform.

A. **HOG Group Threshold Testing**

The main objective of the HOG threshold testing was to determine optimal settings for the grouping threshold (feature constraint) for accurately detecting objects in the field of view. Four threshold tests were conducted and analysis was performed on approximately 1700 frames for each test to check detection rate and the number of feature sets generated on a single object. With a threshold of 1 the HOG function had a hit rate of 80.88% or 1 hit per 1.236 frames. The hit rate is acceptable however 15% of the overall detections were of feature sets consisting of 2 and 3 sets. Although the Track Analyzer component clusters it is still possible for a feature set to be beyond the clustering radius. This can result in the same object being evaluated twice and a potential for model information to be lost depending which set is selected to be the current representation of the tracked model. A value of 2 was found to be the optimal setting when running the system. It had a hit rate of 91.94% or 1 hit per 1.087 frames with only 5.94% being of features sets 2 or greater, a 62.45% reduction from 1. Values of 4 or greater are too constrictive and caused issues with coordinate tracking updates and prediction. Value 4 had a hit rate of 25.41% or 1 hit in 3.934 frames with 4% of the hits being of feature sets of 2 or greater. Given the velocity of an object when detected it possible that the Track Analyzer will not be able accurately maintain coordinate information on the track resulting in a larger discrimination area to be needed to detect its presence. Value 6 was the highest value tested and was the least responsive when detecting an object. It had a hit rate of .3% or 1 hit per 29 frames with .117% being feature sets greater than 2. No doubt an issue when generating accurate coordinate data for tracking, there is also a greater chance of constant drop track conditions when utilizing values greater than 6. Note that a value of 0 represents a disabled threshold and therefore was not tested due to the randomness of the data it generated (Reference Table 1 in Appendix E).

B. **HOG SVM Image Processing**

Testing was conducted on the HOG’s SVM using images of varying quality. Using different bit rates of 64Kbps focusing on motion smoothness and 128Kbps focusing on image quality, two tests of approximately 1000 frames were used on each configuration to determine the impact the sharpness of an image would have on HOG’s SVM detection method for a single object. A PTZ configuration utilizing motion smoothness produced a detection rate falling to less than 50%. Images generated under this configuration output objects that were distorted enough to prevent the visual encoding process from producing accurate feature data prior to being processed by the SVM. Features were either blurred or blended into other elements within the image making the target’s features indistinguishable. In contrast, the camera in a configuration focusing on image quality had a 79.6% increase in object detections. The sharpness of images does have a direct impact on feature generation and object detection (Reference Tables 2 and 3 in Appendix E).

C. **ROI Pixel Area vs. Tracked Model Pixel Area**

The Track Analyzer’s post processing of HOG Processor data was critical when evaluating the prototype systems’ ability to extract model (feature area) information and track subjects. Once the feature areas or ROIs are handed off to the Track Analyzer for post processing from the HOG Processor two possible outcomes are possible prior to filtration or discrimination. The first is no change to the feature area. No modifications to the feature area indicate that the detected object did not exceed any boundary of the image. The second outcome is a reduction in the feature area based on the number of boundaries of the image it exceeds. Using a 320 x 240 resolution and an object moving randomly in the field of view approximately 8 to 10 feet away from the camera, the HOG Processors’ detection area was compared to its’ Track Analyzer’s post processed counterpart to determine if the feature area reduced or remained the same size. 1400 pre and post feature areas were collected and compared. All post processed areas were equal to or less than the HOG generated areas. HOG had an average area of 17900 pixels squared versus 15300 pixels squared processed by the Track Analyzer. Over the course of run the average data loss was 14.5% (Reference Figure 1 in Appendix E). Data loss is always to be expected when an object approaches an image edge. Additionally, the discrimination radius aids in counteracting the feature area reducing mechanism to allow the object be found during tracking.

D. **GSL Prediction vs. Last Known Position**

Using the coordinates of the HOG function as truth data, a comparison of how the system tracks in regards to using GSL prediction based on statistical information and last known object position was performed to assess how accurately the system could track a mobile object. Various search radii were used to confirm hits and miss for each tracking method. As stated in section IV, last known position was used to compensate for GSL weighted bin behavioral issues. Approximately 650
data points with a pixel variance of 15 (parameter to determine when last known position supersedes GSL) were taken from a system test to demonstrate tracking functionality involving a mobile object. Immediately it is apparent referencing the Table 4 in Appendix E that the last known position can more accurately track a mobile object than the predicted position. The inconsistency is based on when the object becomes stationary momentarily. An object in a relatively static position will cause the GSL bin with associated coordinate value to rise quickly and become the more dominate predicted point in the image. Once an object begins the move the Track Analyzer component will assume that the track is still within the previous area and will continue to do so until, a drop track condition is met, the object returns to the vicinity of the predicted position, or the object stops again causing the dominate point in the GSL change based on last known position updates. A last known position aids in keeping the system in check when tracking an object. Although increasing the discrimination radius aids in locating an object, it is not ideal since multiple objects can be detected leading to an increase in overhead processing. Determining when to use last known position versus when to use the predicted coordinates is based on user defined variance between the two points. The Track Analyzer used 108 predicted points (16.438%) and 549 last known points (83.561%) during the test. Options to correct the behavior of the GSL histogram functions are to either purge the histograms if the variance between the predicted and last known positions varies greater than X number of times and create a new set of histograms with the last known coordinates with the starting position or shift and/or scale the histograms based on current position data.

E. GSL Prediction On Confined Movements

A limited movement area test was conducted (simulating a speaker at a podium) to evaluate the accuracy of predicted coordinates. With the same pixel variance parameter as in the mobile object test the predicted value achieved 93.29% of the overall values used to track the object (Reference Table 5 in Appendix E). Both tracking tests demonstrate the limitations in the GSL’s ability to locate moving objects. Objects in a constant moving state in the environment or entering the field of view with a high velocity will result in inaccuracies when GSL attempts to predict the targets location. GSL does however track effectively with objects limiting themselves to movement in a particular region of the image.

F. Test Hardware Platform I/O Performance

Runtime I/O data was collected to gauge the test hardware platforms capabilities. Appendix E’s I/O Testing table captures two minutes of I/O testing for idle and runtime state configurations. The trend of the CPU usage among the four cores with four cameras, the test hardware platform could in theory support potentially a nine camera configuration to maximize all cores on the system. With 3GB of available memory a four camera setup used approximately .2GB of RAM (approximately 200MB) or .4GB when possibly attempting to max out the system. The greatest memory consumption was observed with the cache and buffer. The memory cache increased 120MB and the buffer doubled from an idle state when running a 4 camera configuration. Another item of interest was the number of inbound packets received initially by the system when the prototype began execution. Each time a camera was added to the system, the surge of packets increased. From a 1 to 4 camera configuration the initial flow packets increased nine times that of a single camera (Reference Table 5 in Appendix E). This behavior at times caused a minor delay in frame processing and has been observed to generate corrupt data that is processed by the system (pixilated frames). However, it should be noted that the system would eventually stabilize after the initial packet burst. This item of interest is under investigation to determine the root cause.

VIII. FUTURE WORK AND CONCLUSION

This project was designed to exceed the modeling and tracking capabilities of the Capture Agents used by the Matterhorn and similar visual tracking systems. It has demonstrated a methodology to implement a multiple camera tracking system focusing on filtration, discrimination, and data sharing methods allowing each camera to act as its own director and cameraman unit. This research has also presented items of interest that need to be taken into consideration when developing this type of system. These include unforeseen behaviors in hardware in terms of autonomous actions and abrupt failures and how to counteract or mitigate these issues in the system. The hardware components can actually increase or decrease the effectiveness of the system. A lessor quality setup can yield poor results since settings have to be configured to a lower standard to handle all of the necessary processing. In contrast, third party software needs be taken into account. It is necessary to thoroughly integrate and test any software that will be a part of the system and determine how it will influence the systems’ functionality. A desired outcome of this research was to demonstrate a baseline approach for implementing an efficient multiple camera tracking system with limited resources as well as an educational reference to derive similar systems. Future projects derived from this could include visual cueing (trigger the system to focus the subject that is associated with
the cue object), corrupt frame mitigation (assessing how much of the frame is acceptable for system processing), Matterhorn integration (integrate the prototype system with multiple cameras into a Capture Agent), or improve the statistical capabilities for data prediction for GSL. Additionally given the fact that data is sent over the network is in plain text, it also a possibility to implement an encryption mechanism that allows secured communications between the prototype system and the PTZ cameras.

VIII. REFERENCES

[1] Cha Zhang, Yong Rui, Li-wei He, Hybrid Speaker and Michael Wallick, “Tracking in An Automated Lecture Room”


[19] Ethan Rublee Vincent Rabaud Kurt Konolige and Gary Bradski, “ORB: an efficient alternative to SIFT or SURF”


[33] Nicole M. Artner, “A Comparison of Mean Shift Tracking Methods,” Digital Media, Upper Austria University of Applied Sciences, Hagenberg, Austria

[34] R. Stolkin, I. Florescu, G. Kamberov, “An Adaptive Background Model for Camshift Tracking with a Moving Camera,” Dept. of Computer Science Stevens Institute of Technology


[38] Qiang Zhu, Shai Avidan2, Mei-Chen Yeh, and Kwang-Ting Cheng, “Fast Human Detection Using a Cascade of Histograms of Oriented Gradients,” 1Electrical & Computer Engineering Department University of California at Santa Barbara, 2006


APPENDIX A

Major Project Components:

- Ubuntu 11.04 – Natty Narwhal
- OpenCV 2.2.0
- cURL 7.3
- Xerces 3.1.1
- GSL 1.16
- ZyXEL IPC4605N Camera
- D-Link 5020L Camera

OpenCV support libraries order of install:

- build-essential
- libgtk2.0-dev
- libjpeg62-dev
- libtiff4-dev
- libjasper-dev
- libopenexr-dev
- cmake
- python-dev
- python-numpy
- libtbv-dev
- libeigen2-dev
- yasm
- libfaac-dev
- libopencore-amrnb-dev
- libopencore-amrwb-dev
- libtheora-dev
- libvorbis-dev
- libxvidcore-dev
- ffmpeg

**APPENDIX B**

System Configuration Parameters:

- \texttt{fz\_id} – Unique numerical ID for the network camera. Used in accessing data that is camera specific.
- \texttt{control\_proto} – Internet protocol used to control the network camera (http, etc)
- \texttt{stream\_proto} – Internet protocol used to get the network camera image stream (rtsp, etc)
- \texttt{cam\_user} – Username for network camera access
- \texttt{cam\_passwd} – Password for network camera access
- \texttt{cam\_ip} – The IP address needed access the network camera
- \texttt{cam\_stream\_port} – Stream port of the image stream
- \texttt{cam\_control\_port} – Server port on the network camera used to send camera control commands
- \texttt{cam\_stream\_url} – URL appended to the stream protocol string
- \texttt{strm\_type} – 0 for MJPEG and 1 for H.264 or MPEG4
- \texttt{cam\_res\_height} – Network camera resolution height
- \texttt{cam\_res\_width} – Network camera resolution width
- \texttt{cam\_up\_command} – Query string to move PTZ camera up
- \texttt{cam\_right\_up\_command} – Query string to move PTZ camera up and to the right
- \texttt{cam\_right\_down\_command} – Query string to move PTZ camera down and to the right
- \texttt{cam\_right\_command} – Query string to move PTZ camera right
- \texttt{cam\_left\_up\_command} – Query string to move PTZ camera up and to the left
- \texttt{cam\_left\_down\_command} – Query string to move PTZ camera down and to the left
- \texttt{cam\_left\_command} – Query string to move PTZ camera left
- \texttt{cam\_down\_command} – Query string to move PTZ camera down
- \texttt{cam\_zoom\_command} – Query string to zoom the PTZ camera
- \texttt{cam\_home\_command} – Query string to move PTZ camera to the home position
- \texttt{confine\_tpl\_x} – Define upper left x coordinate of the confinement zone
- \texttt{confine\_tpl\_y} – Define upper left y coordinate of the confinement zone
- \texttt{confine\_br\_x} – Define lower right x coordinate of the confinement zone
- \texttt{confine\_br\_y} – Define lower right y coordinate of the confinement zone
- \texttt{detector\_method} – Select the HOG detector method to be used
- \texttt{bound\_rect\_color} – Select color of the bounding rectangle to be used on object detected by the detector
- \texttt{cam\_command\_str} – A variable set to the NULL string to initialize the first query string
- \texttt{time\_to\_pan} – Number of seconds an object can be outside the confinement before a pan operation is executed
- \texttt{time\_to\_tilt} – Number of seconds an object can be outside the confinement before a tilt operation is executed
- \texttt{time\_to\_drop} – Number of seconds an object can not be tracked before the system drops the track and reset
- \texttt{max\_obj\_tracks} – Number tracks the system can track at once
- \texttt{cluster\_radius} – Distance from a center point that bounding rectangles can be grouped together
- \texttt{discrim\_radius} – Distance from a predicted point that bounding rectangles could be considered a potential track
- \texttt{percent\_match} – Initial filtering discrimination percentage. This parameter is used to determine which objects are available for a camera to track.
- \texttt{has\_track} – Always set to “false.” This parameter tells the system that the camera has no track when the system initializes.
- \texttt{inline\_adjust} – Adjust the center point in the bounding rectangle
- \texttt{cc\_perent} – drop track percentage used to determine when cameras may have the same tracked object
- \texttt{fr\_wait} – number of frames to skip when a camera movement occurs
- \texttt{pos\_thrsh} – keep last known position of object if predicted coordinates if the variance between current position and predicted position exceeds this value

**APPENDIX C**
List of configuration and source files:

- *build_cam_system.sh* – Script file that builds the tracking prototype system.
- *cam_system.xml* – Contains the configuration information for each PTZ attached to the system.
- *camera_controller.cpp* – C++ class used by the system to control and query images from a PTZ camera.
- *camera_controller.h* – C++ header file for the camera_controller object
- *fast_match_template.cpp* – Open code C++ code that provides the FastMatchTemplate function
- *fast_match_template.h* – Open Source C++ header file for the FastMatchTemplate function
- *global_params.h* – C++ header file that contains all libraries and data structures common to classes used by the system to process image and timing data.
- *hog_processor.cpp* – C++ class used by the system to obtain information from OpenCVs’ HOG implementation (Bounding Rectangles) with associated time stamping.
- *hog_processor.h* – C++ header file for the hog_processor object.

- *main.cpp* – Function main()
- *param_loader.cpp* – C++ class used by the system to load the configuration parameters from the cam_system.xml file
- *param_loader.h* – C++ header file for the param_loader object
- *track_analyzer.cpp* – C++ class used by the system to control and analyze data from queried images.
- *track_analyzer.h* – C++ header file for the track_analyzer object
- *xmldom.cpp* – Open source C++ class used by the system to parse XML files.
- *xmldom.h* – Open source C++ header file for the xmldom object.

APPENDIX D

System Component Flow Charts:
Prototype System High Level Execution Flow
HOG Processor Execution Flow
Track Analyzer Execution Flow Part 1
Track Analyzer Execution Flow Part 2

A

- Start Track Timer
- Generate ROI Image
- Update Track Fuzer With ROI
- Set PTZ Camera Tracking Status To True
- Receive HOG Object Tracking Structures
  - Update X and Y Position Histograms
  - Pan, Tilt, Or Zoom Criteria Met?
    - Yes:
      - Adjust Camera
    - No:
      - Cross Check Model

B

- Predict X And Y ROI Position
- Possible Models Found?
  - No:
    - Receive HOG Object Tracking Structures
  - Yes:
    - Fast Match Template
    - Select Model With Highest Percentage Match
    - Update ROI Tracked Image
    - Update Track Fuzer With ROI
    - Drop Track?
      - Yes:
        - Set PTZ Camera Tracking Status To False
      - No:
APPENDIX E

Table 1: HOG Constraint (Group Threshold) Statistics

<table>
<thead>
<tr>
<th>Feature Sets</th>
<th>Constraint Value</th>
<th>1 (Least Restrictive)</th>
<th>2</th>
<th>4</th>
<th>6 (Most Restrictive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>693</td>
<td>318</td>
<td>1487</td>
<td>1667</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1106</td>
<td>1465</td>
<td>364</td>
<td>56</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>242</td>
<td>98</td>
<td>68</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>27</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: HOG Detection (Low Quality Images) Statistics

<table>
<thead>
<tr>
<th>64 Kbps (Motion Smoothness)</th>
<th>Hits</th>
<th>Misses</th>
<th>Hit %</th>
<th>Miss %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>491</td>
<td>566</td>
<td>46.5</td>
<td>53.55</td>
</tr>
</tbody>
</table>

Table 3: HOG Detection (High Quality Images) Statistics

<table>
<thead>
<tr>
<th>128 Kbps (Image Quality)</th>
<th>Hits</th>
<th>Misses</th>
<th>Hit %</th>
<th>Miss %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>882</td>
<td>175</td>
<td>83.4</td>
<td>16.56</td>
</tr>
</tbody>
</table>

Figure 1: HOG Feature Area vs. Track Analyzer Post Processing Mechanism

Table 4: Object Coordinate Prediction and Tracking Statistics

<table>
<thead>
<tr>
<th>Image Resolution 320x240</th>
<th>HOG vs PREDX</th>
<th>HOG vs LKX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hits</td>
<td>Misses</td>
</tr>
<tr>
<td>Discrim Radius: 10</td>
<td>88</td>
<td>568</td>
</tr>
<tr>
<td>Discrim Radius: 20</td>
<td>133</td>
<td>523</td>
</tr>
<tr>
<td>Discrim Radius: 30</td>
<td>149</td>
<td>507</td>
</tr>
<tr>
<td>Discrim Radius: 40</td>
<td>179</td>
<td>477</td>
</tr>
<tr>
<td>Discrim Radius: 50</td>
<td>236</td>
<td>420</td>
</tr>
<tr>
<td>Discrim Radius: 60</td>
<td>310</td>
<td>346</td>
</tr>
<tr>
<td>Discrim Radius: 70</td>
<td>394</td>
<td>262</td>
</tr>
<tr>
<td>Discrim Radius: 80</td>
<td>443</td>
<td>213</td>
</tr>
</tbody>
</table>
### Table 5: Confined Object Movement Statistics

<table>
<thead>
<tr>
<th>Discrim Radius:</th>
<th>10</th>
<th>HOG vs PREDX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Hits</td>
<td>324</td>
<td>Total Misses</td>
</tr>
<tr>
<td>Total Misses</td>
<td>64</td>
<td>%Hit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%Miss</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.8350</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.1649</td>
</tr>
</tbody>
</table>

### Table 6: I/O Testing (Monitoring memory, CPU, and network) Statistics

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Idle</th>
<th>1 Camera</th>
<th>2 Cameras</th>
<th>4 Cameras</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max CPU (4 Cores)</td>
<td>6%</td>
<td>80%</td>
<td>142%</td>
<td>181%</td>
</tr>
<tr>
<td>Main Mem(4GB)</td>
<td>3GB</td>
<td>.1GB</td>
<td>.1GB</td>
<td>.2GB</td>
</tr>
<tr>
<td>Mem Cache</td>
<td>290.2MB</td>
<td>313.2MB</td>
<td>384MB</td>
<td>408.1MB</td>
</tr>
<tr>
<td>Mem Buffer</td>
<td>37.6MB</td>
<td>38MB</td>
<td>39.1MB</td>
<td>81.2MB</td>
</tr>
<tr>
<td>PCKT In Initial</td>
<td>N/A</td>
<td>832</td>
<td>3205</td>
<td>7277</td>
</tr>
<tr>
<td>PCKT In</td>
<td>N/A</td>
<td>1985</td>
<td>5244</td>
<td>11735</td>
</tr>
<tr>
<td>PCK Out</td>
<td>N/A</td>
<td>65</td>
<td>887</td>
<td>3230</td>
</tr>
<tr>
<td>PCK Total</td>
<td>N/A</td>
<td>2050</td>
<td>6131</td>
<td>14965</td>
</tr>
</tbody>
</table>