Distributed Monitoring of Crowds from CCTV

Preliminary Project Proposal

Presented to the

Queensland University of Technology
Science and Engineering Faculty
School of Electrical Engineering and Computer Science
In Partial Fulfilment of the Requirements
for the Degree of Bachelor of Engineering (EN01) (Honours)
(Computer and Software Systems)
By
Wafi Mohammad Hossain
Under Supervision of
Prof. Simon Denman

TABLE OF CONTENTS

I. Introduction

QUT is currently working with a number of industry partners in deploying techniques in a live environment to monitor crowd size, dwell time, and throughput rates; which are aggregated by a web server to display current and historic crowd information to users. This project will investigate the development, testing and deployment of new analytics and/or the development of web based visualizations to explore the data.
Crowd surveillance is crucial as it can help prevent possible threats and large scale incidents. The size of a crowd in a certain place, that is abnormal and different to what is usually expected can easily be a sign of danger. Someone waiting for too long at a kiosk may indicate sinister motives or threat. Abnormal flow of pedestrian at certain point can indicate that people are trying to run from something dangerous. Hence, these sorts of algorithms are vital in providing adequate security.
Currently a live system is deployed in the Gold Coast Airport, and the focus of this project will be dedicated to such system. The main focus of this project is in improving current inefficient algorithms within the system, implement new crucial features and improve existing ones.
Currently emphasis is put into implementing a Support Vector Machine driven algorithm to estimate kiosk dwell times. The current algorithm is not as efficient as it could be. This is to overcome current limitations such as noise and inaccuracies that maybe caused by the un-predictability of people's positioning. Support Vector Machine is a type of discriminative classifier that labels data on a hyperplane. The rationale behind it is to use features to train the Support Vector Machine for kiosks when they are occupied vs when they are not occupied. This will allow the Support Vector Machine to then predict whether a frame is occupied or not on its own. The following analysis explores limitations and downfalls of the developed prototype and how it can be improved for future implementation and deployment. It also assesses feasibility and successes achieved.
The following report, outlines current progress and deliverables achieved during the first half of this project. This includes the updated Literature Review.

I. Literature Review

Following is an updated review of relevant literature that is to help to provide a better understanding of methodologies that are already implemented within the system. This review has been slightly expanded to include further information gathered during the progress of this project. Further expansion and revision is to be expected as the project progresses and further documentation is produced

A. Crowd Estimation Technique: Scene invariant multi camera crowd counting

Crowd counting algorithm is essential for security. Every year a number of incidents related to crowd and mobs happen throughout the world [1], [2]. The following review will focus on [3]. This paper is very important as it details the algorithms and methodologies that are already implemented within the project for crowd counting purposes. The majority of implementations for crowd counting and estimation, focus on single viewpoint and are designed to work within a fixed environment, needing a fair amount of training. In recent years approaches have been made toward more independent systems. The Scene invariant multi camera crowd counting approach used within this project is widely discussed in [3].

This paper discusses approaches that enables the system to be deployed in a range of different settings with minimal training. It also covers, methods to account for the overlap between different cameras, as previous implementations mostly focused on single camera settings. The system is trained on a bank of "reference viewpoints" before being deployed on any number of unseen viewpoints, without any additional training requirements. The approach is based on camera calibration, of which the algorithm is originally published in [4, p. 21].

Scene invariance is achieved through the extraction of features and are then used for normalizing for camera position and orientation. A density map is produced which is used to scale the extracted features. The density map will contain weighting of each pixel at a given location to account for the distance from the camera. In this case this density map is used to normalize for different scenes [3]. The density map model is authored by Tsai and it is also the most accessible within this context [5], [6].

The density map discussed in [3], works by means of a 3D system, where a 3D cylinder of a fixed size is project onto the scene at every pixel representing a location. This cylinder represents the approximate size of a human. The circles are approximated by using a polygon of 20 sides. The notation $R_{i,j}$ is used to represent the cylinder model centered around pixel (i, j), and $|R_{i,j}|$ represents the area of this template in the image plane. Hence the density map is constructed using,

$$S(i,j) = \frac{1}{|R_{i,j}|}$$

Hence a density map is constructed with locations that may be farther away from the camera, representing an object that occupies less pixels, but having higher weight in the density map to account for it, and vice versa for locations that are closer to the camera.

For the approach discussed in [3], background modelling is essential as it allows subsequent steps, such as foreground detection, group segmentation and feature extraction. The methods used to generate a foreground binary mask is described in [7]–[9]. In this context for each foreground 'blob' an estimate size of the crowd is obtained. The total estimates for all of the foreground 'blobs' are added together to provide a full estimation. For training purposes, straight forward ground truth annotation is performed by directly labelling the number of people in a blob. This procedure is further discussed in [4].

The next step is feature extraction to group the segmentation, where each feature is weighted through the density map in order to achieve normalization across the scene. The main feature categories are:

- Size
- Shape
- Edges
- Keypoints

For this method to be ported over to a broader multiple camera setup, overlapping regions are to be accounted for. To do this the following method is proposed in [3]:

An overlap map is produced, denoting how much an object centered around a pixel is visible within other viewpoints. Such Overlap Map can be produced either via a Density Map Modification or via Pixel Density Assignment. Following is a brief description for both methods.

- Density Map Modification:

  Overlap regions are given a lower weight to account for double ups within such region. Hence counting is performed as subsequent step
- Pixel Density Assignment:

  This approach leaves the density ma unaltered, so that the system generally operates in the same manner. Instead crowd densities are modified after counting has been performed, on a pixelwise basis.

B. Large scale monitoring of crowds and building utilisation: A new database and distributed approach

Following is a review of [10]. This is a very essential paper covering the principles of applying existing algorithms, and the methodologies discussed in the previous section to a broader and larger scale for real world settings. Our focus in particular, due to the context and scope of this project, will be on the virtual gate throughput algorithms.

As previously stated a lot of the recent body of work has been done on single camera settings to measure crowd size [3], [11], and monitoring pedestrian flow [12]–[14].

This paper Presents the potential of a large database obtained from 12 cameras, placed in and out of a university building. Such database is then used to implement video analytics such as crowd counting and virtual gate throughput estimation, hence enabling further possibility within a field that has limited resources in terms of large databases and large scale applications.

In terms of scalability the structure of the cameras used need to be portable over to larger scale systems. To achieve that each camera is assigned its own virtual machine, so that in the future all the data can be then assimilated into a main framework for finalizing the analysis.

The crowd counting methods implemented in this papers is reviewed in the previous section from [3].

In terms of virtual gate throughput estimation, various approaches have been previously taken. Such approaches include optical flow perpendicular to a single line in a scene [13], segmented entry/exit events through a 2d virtual gate[14] and using local HOG features and integer programming[12].

The paper itself uses a local feature [10] approach analogue to the crowd counting method discussed in [3]. This method implements a counting line surrounded by a region of interest (ROI) and direction of interest (DOI). R represents the set of pixels belonging to the ROI and the unit vector pointing in the DOI is denoted d. In the proposed algorithm, optical flow is accumulated within the DOI at a set of points. These points are determined using a feature point selection criteria. Hence, each video sequence is divided into a set of sub-sequences where optical flow is accumulated and regression is performed to estimate the number of people passing through such virtual gate.

A three feature point selection criteria is used where all pixels within the ROI are feature points, edges are detected using Canny edge detection[15], and FAST is used to locate a set of corners all within the ROI [16].

The following equation represents the optical flow v_t at a time t at pixel P,

$$\widehat{v_t}(P) = \widehat{v_t}(P) \cdot d$$

Where d is the direction where this flow is headed or simply called aligned optical flow. Following up at each frame for the total aligned flow we have,

$$a_{t,f} = \sum_{P \epsilon F_{t,j}} \widehat{v_t}(P)$$

Where $F_{t,j}$ is a set of featured detected within the ROI at time t, where f is type of feature under consideration (all pixels, edges, or corners). Hence, across each time window W_n (A subset video sequence) the total aligned flow is accumulated,

$$\alpha_{n,f} = \sum_{t \epsilon W_n} a_{t,f}$$

Hence this summation is roughly proportional to the number of feature points crossing the counting line.

Optical flow histograms are used to separate the effects of potential noise and true motion [10].

Basically, to account for noise aligned optical flow is calculated within different histogram bins as follows:

Each pixel P is assigned to a histogram b based on the magnitude of its optical flow. The set of all features and histogram bins are collected into a feature vector $x_n$which describes the time window W_n. A regression model (Gaussian Process Regression)[17] is then trained to learn the relationship between the feature vectors and the ground truth values. For each window, a positive and a negative feature vector is obtained. Hence, by training two regression models bidirectional flow tracking is obtained.

C.Support Vector Machine

Following is a brief investigation into current support vector machines available. This brief overview will focus on applications within computer vision sectors. Such overview is to give a basic idea of what has been done and what's possible with Support Vector Machines.

In [18] it is discussed how various challenges such as moving backgrounds, cluttering, scale variation etc. have been addressed in computer vision in the past in the field of human motion detection [19]. Various methods have also been proposed for recognizing human actions directly from image measurements [20]. However, all previous methods are very prone to external factors interfering with the algorithm, and thus a vast portion of these algorithms scalability is not ideal. Hence in [21], a method is proposed where local features from special recognition are fed into an SVM in a robust classification approach. In a similar manner an approach is proposed by using this method for human action recognition in [18].

Within the field of visual category recognition, a hybrid solution with nearest neighbour classifiers and support vector machines is proposed in [22]. In this approach, the high variance of neighbour classifiers and the demanding computational and training load of SVMs are mitigated by the combination of both methods.

Within the current context it is known that the data used to test or implement the SVM can be very large. [23] presents a good example of large scale image classification. This is the first investigation that dives into large scale application purely because of the lack of public large scale databases for such purpose. SVM have also been used in embedded solutions for human action recognition [24].

II. CURRENT PROGRESS AND METHODOLOGIES

A.Improved Kiosk Dwell Time

In the current stage of the project, most of the focus is directed towards implementing an appropriate algorithm to measure kiosk dwell time for the various kiosks that are being monitored. This approach is to replace the existing rudimentary implementation for improved efficiency and precision.

To do this, the prescribed approach is to use Support Vector Machines. A Support Vector Machine is basically a discriminative classifier. Given labelled training data, the algorithm outputs an optimal hyperplane which categorizes new examples [25].

Hence this concept is to be applied to kiosk dwell time estimation. For each kiosk there is to be a region of interest (ROI). Such ROI is to be split into subsections for which features are to be extracted. Hence the an SVM is trained using those extracted features. After appropriate training for a given kiosk, the SVM can start estimating whether or not a certain kiosk is occupied.

1) Current Progress

a) Working environment

Most of the development is conducted on an ASUS UX305UA Ultrabook which houses a core i7-6500U quadcore processor at 2.5Ghz turbo boost up to 2.59. it houses 8gb of ram and 256 of SSD. Features are mostly developed and tested within an Ubuntu Virtual box ran on this machine natively on Windows 10.

Libraries used for development of back end features include OpenCV, VXL and other libraries produced by academics at QUT.

b) Simple two image demo classification

Initially, for the author to get familiarized with the basic concepts and how an SVM generally works a very simple demo algorithm is implemented within the environment. This code is given in Appendix A. This implementation is a very simplified, one class SVM that takes in only one feature and identifies it within the hyperplane against anything that differs from that feature. Within the context of the project, this type of approach may not be ideal, but it is a starting point for grasping the core concepts. This one class SVM is first trained with the RGB values from the image of a sample kiosk. Generally speaking, a pixel model is extracted from such image, where from each pixel, the RGB values are stored in a two-dimensional array. Each row is to represent a channel and each column is to represent a pixel from the image. Hence the SVM is trained with such pixel model.

Figure 1 – Sample kiosk image to produce training pixel model

As shown in figure 1 this kiosk picture image is of a very limited colour range. This sample is chosen on purpose since for our simple test to work, a picture with a quite diverse range of colours is to be tested from the ground truth presented in figure 1.

Figure 2 – Test "occupied" kiosk image

Figure 2 shows a modified image of figure 1 that is going to be used to test the SVM's functionality. As shown in figure 2, the sample image possesses a fair amount of colour variation to allow for the SVM to detect differences. Hence, after the SVM is trained, the test code will then create a pixel model of the image shown in figure 2 and fetch it to the SVM to predict whether or not this test image is occupied. If less than 80% of the pixels are classified as being part of the ground truth image, then the test image is occupied.

c) SVM Trainer Prototype Updated

After careful preparation, evaluation and meeting with the supervisor of this project, a sample skeleton code is laid out to implement a SVM trainer algorithm.

The aim of this algorithm is to train an SVM with ground images, where, from a given video from the gold coast database is analysed and a set of frames are explicitly annotated as being ground trut for a given kiosk.

Hence, these frames are passed through the crowd counting algorithm and a feature is extracted for each subregion within the ROI of the respective frame. All the features extracted, both for ground truth frames, and frames where the kiosk is occupied are added into a vector; and then, another vector is to be generated labelling which features are positives (empty ground truth frames) and which features are negative (occupied frames). All of this is then fetched into the SVM for training purposes.

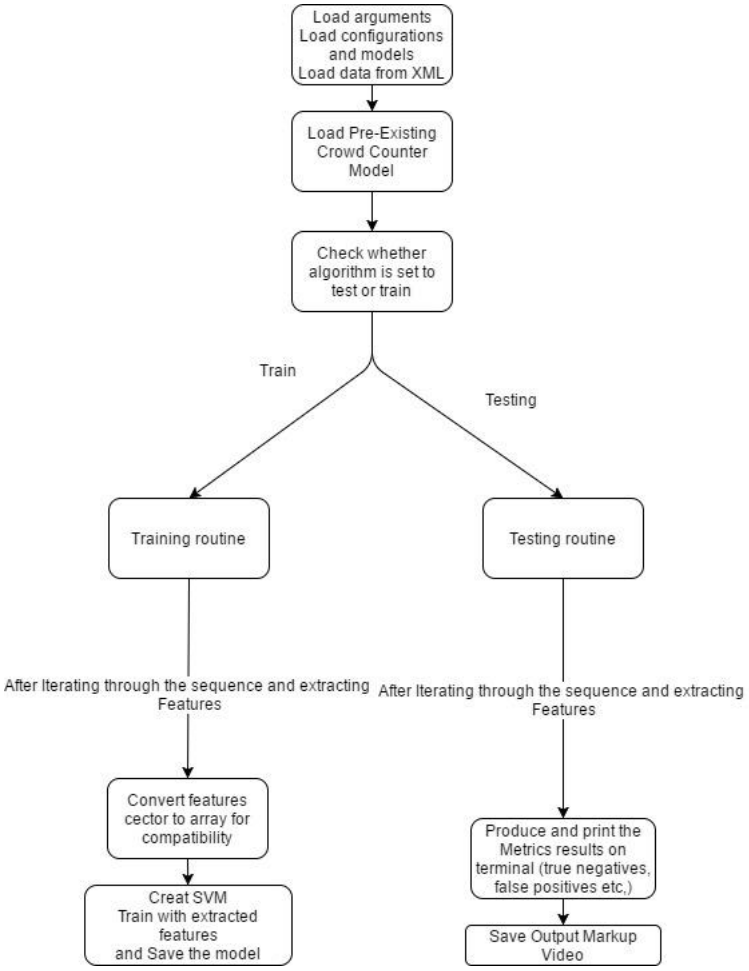

Figure 3 – Flow chart of the SVM prototype

Most of the relevant information to set up the algorithm is contained in an Extensible Markup Language (XML). XML is originally developed for ease of implementation and use[26]. In this context, such file is used to configure the algorithm, load pretrained models, coordinates, and other relevant information.

The initial for loop, first counts how many kiosks data are presented in the XML file and hence it stores the respective coordinates for the region of interest (ROI) in a vector. The second for loop loads the notations for the occupied events within the vector for the respective ROIs. Hence the third for loop in turn is to load the test events notation which are the occupied events notation for a different section of the video than the training section.

Testing events are meant to be used to produce metrics and efficiency data to assess the algorithms efficacy and success. Following is an abridged simplified version of the XML snippet that is usually used to configure the algorithm:

```
<KioskData>
    <qut-video-normal path="/home/webmaster/saivt-vxl/qut/qutapps/qutSECAnalytics/Data/9996-640x480.avi" />
    <qut-video-motion path="/home/webmaster/saivt-vxl/qut/qutapps/qutSECAnalytics/Data/9996-640x480-motion.avi" />
    <SavePath path="/home/webmaster/saivt-vxl/qut/qutapps/qutSECAnalytics/models"/>
    <TestFramesStart start="15021" end="22522" />
    <frames-used frames="15021" />
    <Divisions Hdiv="10" Wdiv="10"/>
<Kiosk>
<ROI>
<point x="122" y="188"/>
<dimensions height="96" width="31"/>
</ROI>
        <Event start="135" end="147" />
        <Event start="541" end="695" />

        <TestEvent start="15021" end="15347" />
        <TestEvent start="15620" end="15644" />
</Kiosk>
```

Figure 4 – Abridged XML code snippet

In figure 4 code snippet, the basic structure of the XML used in this implementation is shown. The parameter "qut-video-normal" has the file path for the normal video. The parameter "qut-video-motion" contains the segmented video filepath. The parameter "SavePath" contains the path where the SVM trained model is to be saved after training. "Kiosk" contains paramaters for a single kiosk. "ROI" contains the coordinates and dimension of the region of interests. Each kiosk will contain various "Event" parameters representing the start and end frame where the kiosk is effectively occupied.These are annotations used to for training purposes. Lastly, "TestEvent" parameters are used for testing. These are annotation used for testing purposes. The SVM will predict whether the kiosk is occupied and this prediction will be compared against the notations.

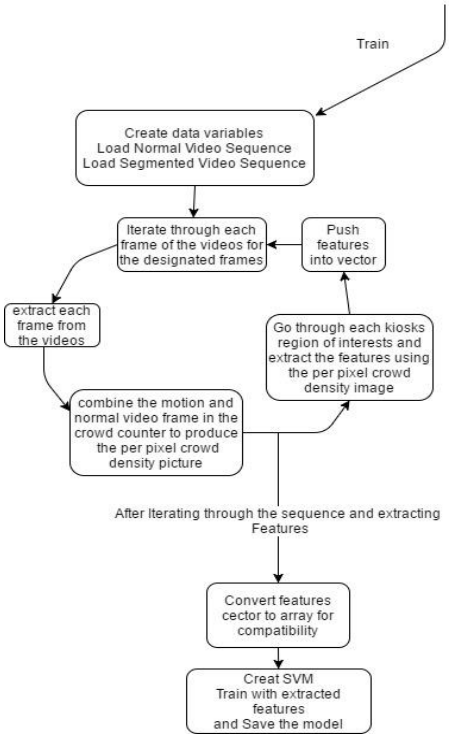

Figure 5 – Training Routine

Refer to lines from 151 to 240 in Appendix B for further details and the specific code that executes the training routine. During each iteration, the current frame for both the motion video and the normal video are processed in the crowd counter to produce the per pixel crowd picture. Hence, the first for loop is used to check if the current frame is occupied, and populate the labels array with the appropriate value. The features are then extracted and pushed on a vector. The way features are extracted has been updated for

Features are extracted by fetching the per pixel crowd density image to the appropriate function. For further details refer to lines from 349 to 394 in Appendix B. The features are extracted by iterating through all the kiosks and hence going through their respective regions of interests. The underlying function is to iterate through the given region of interest and collect features from each pixel. The function is to split the region of interest by a number of subregions, and add the pixel density for each pixel at each subregion. Hence, the values are populated in a vector that will contain the features for that specific kiosk at that specific frame. This is then added to the main features vector. The per pixel crowd is first cropped to the respective ROI. In addition to the pixel densities,

Image orientation are added based on histograms. And the frequency of the orientation of a given pixel is added to the feature vector. This is further discussed in the following sessions.

After all features have been extracted and the iteration is done, vectors are then converted to arrays for compatibility. The labels array containing the appropriate labels (1 for occupied and 0 for not occupied), the array containing the features are used to create and train an SVM model. The SVM model is then saved to the filepath specified by the XML file.

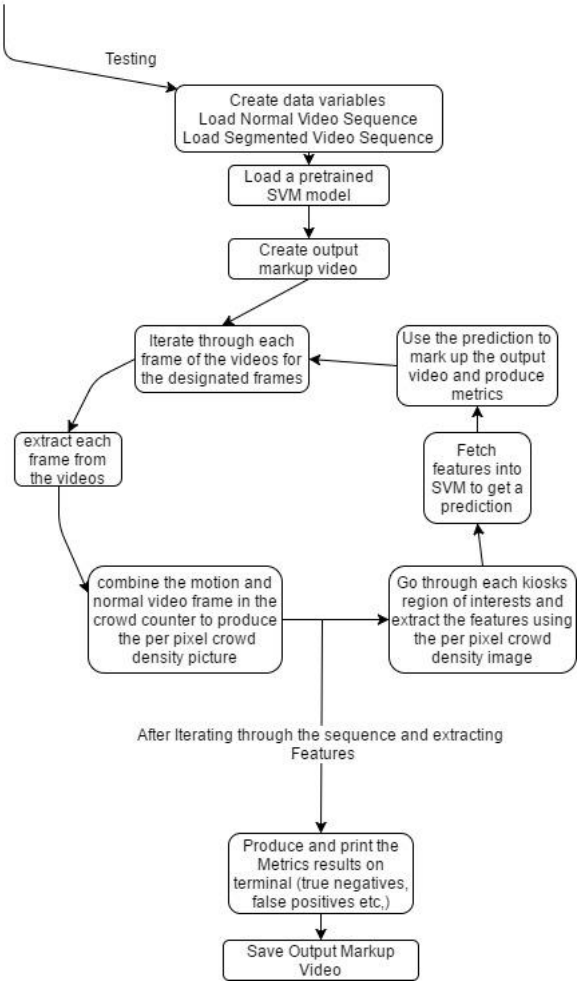

Figure 6 - Testing Routine

Figure 6 shows the current testing and assessing procedures. This is very similar to the previous training procedure. The result of the prediction is stored in a variable. The result is then compared to the expected results from the test events present in the XML file. Based on such comparison, true positives, false positives, true negatives, and false negatives are calculated as a percentage of all the predictions made in the testing routine. A mark-up video is also produced where the kiosks are marked when the SVM predicts that the kiosk is occupied. For the testing routine a different section of the video sequence is used from the training section. For further details check lines from 245 to 353 in Appendix B.

2) *Experimental Findings and Current Progress*

a) Initial experimentation

The following screenshot in figure 7 shows a typical example of the prototype at work. In this case it appears to be behaving appropriately. It is a screenshot taken from the Markup output video. For the following experiments and assessments, the three kiosks on the left from figure 7 are used for the prototype.

Figure 7 - Regular screenshot from markup video

However, there are a lot of cases where the algorithm fails and the predictions are not accurate. Due to this, metrics are produced to get a better understanding of how it fails and the reasons behind it. Lines 307 to 318 from Appendix B are where the metrics are produced and calculated. In other words, true positives, false positives, true negatives and false negatives for each kiosk are counted. Then, from line 344 to 349 in Appendix B, these metrics are printed on the command line as a percentage out of all predictions made.

```
truePositives: 12.75%
trueNegatives 33.77%
falsePositives: 32.15%
falseNegatives: 21.33%
webmaster@webmaster-VirtualBox:~/saivt-vxl/bin/qutapps/qutSECAnalytics$
```

Figure 8 - Screenshot of commandline output for metrics

Figure 8 shows the output of the calculated metrics for the prototype. It can be clearly observed that the prototype produces accurate prediction approximately half of the time. Initially the expected accuracy of the algorithm was around 10%. However, even though the prototype performed better than expected, it is still quite inaccurate.

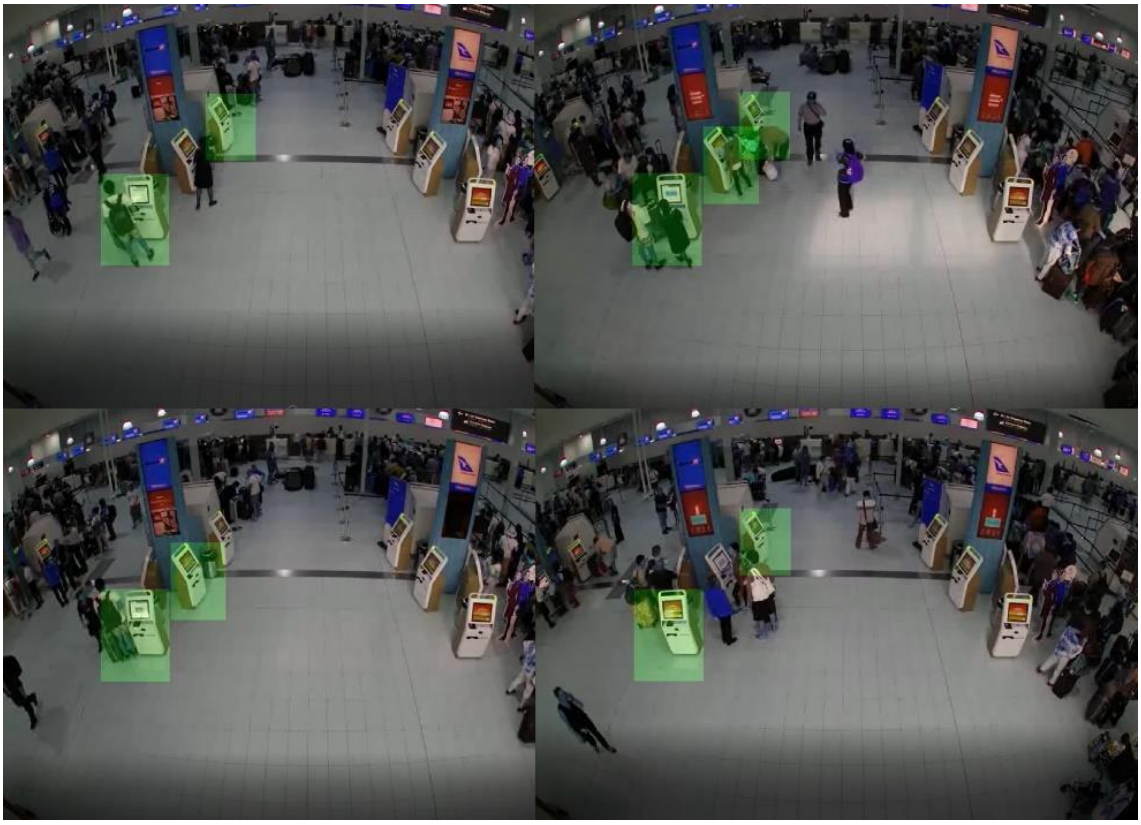

Figure 9 - Various failure cases

Figure 9 shows a few failure cases of the prototype. From the screenshot in the top left corner it can be observed that the person standing in front of the middle kiosk is triggering the prototype to classify the kiosk on top to be occupied. In the top right screen shot the top kiosk is seen as occupied even though the crowd is using only the middle kiosk. In the bottom left screenshot, the middle kiosk is being labelled as occupied even though it is empty. For the bottom right screenshot, the bottom kiosk and the top kiosks are triggered, by crowds that are outside those kiosks. The middle kiosks should be labelled as occupied but it is ignoring the crowd.

Failures such as these can occur for various reasons. The primary reason is the use of only one feature. More features are needed, for the SVM to be able to classify frames more accurately. Such features can include texture, edges and more. Secondly the way each kiosk is annotated as occupied for training, can have a big impact on how the prototype is trained. Hence, annotation for training purposes are to be revised. Lastly it is appropriate to note that due to the angle and location of the kiosks, respective to the camera, noise may be unavoidable. The various regions of interests can also be adjusted to suit the algorithm better, however slight noise may still be present after all the optimizations have been implemented.

a) Further Experimentations and Results

Further testing has been executed, after proper consultation. Initially the events were manually annotated again to further reflect an appropriate behaviour from the prototype's perspective so the machine can learn more appropriately and accurately. This approach has not impacted outcomes in major ways. the output is still approximately correct half of the time.

The size, and placement of the ROI has been further manipulated to observe behaviour of the prototype. In this case only one of the kiosks from the left has been taken into consideration for further testing and simplification of the model. Four different sizes and placements of the ROI have been tested and they are as follows:

- x=100, y=188, height=96, width=57
- x=110, y=188, height=115, width=40
- x=90, y=180, height=130, width=60
- x=122, y=188, height=96, width=31

All of these ROI produced, no significant change in results. The last coordinates and ROI dimensions are kept for simplicity. The approach of working with a single kiosk has been deemed appropriate since the problem is to be approached at a core level first, where smaller parts are analysed first.

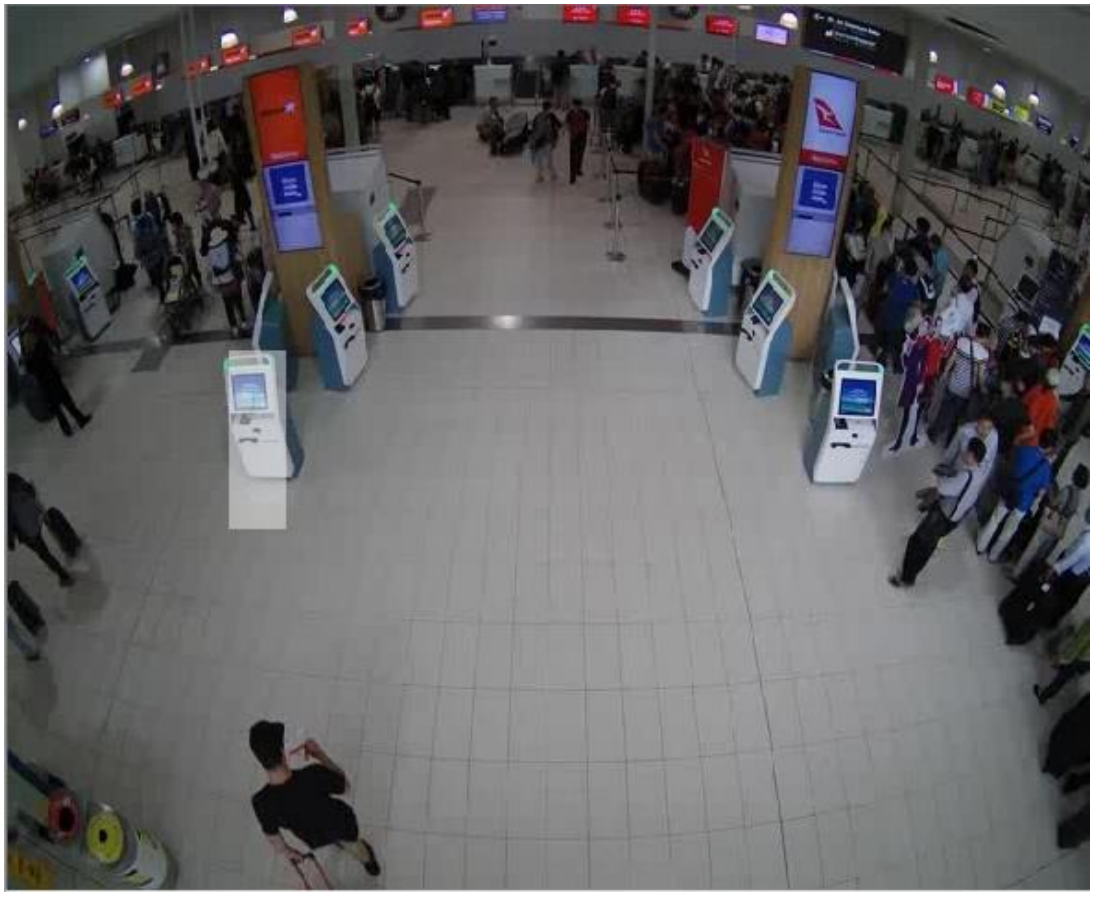

Figure 10 - Current ROI for the relevant kiosk

Figure 10 shows the current ROI used for testing. This has been chosen as the base ROI for a kiosk for future testing.

b) Implementing orientation histogram features

The most recent approach has been to add orientation features to the training data for the support vector machine. This involves producing an image that contains orientation features for each pixel. These orientations are within the range of -180 to 180 respective to the camera. Hence these orientations are divided within bins (ranges). Hence the frequency for each bin the frequency of how many pixels reside within such bin has been calculated and hence fetched into the feature vector for training. Currently this has produced poor results. Initially the orientation for each pixel has been calculated based on the per pixel crowd density image extracted from the current frame. Due to lack of success, the original image for the current frame has been used. All of the predictions come out as 1 (meaning the kiosk has been predicted to be occupied) for all the frames. This behaviour is quite erratic and it needs to be further explored in the upcoming stages.

3) Further experimentation

Different number of histograms have been tried without success. Reannotation has been executed for the video sequence 9996. No progress has been yielded with that methodology.

a) Faulty feature SVM

One major issue discovered during testing has been the faulty training and array assignment for data to be used for the SVM training. This happened due to everything being assigned in arrays and the way such array was converted to a Matrix to be fetched into the SVM. To fix this, arrays and pointers have been eliminated and everything is now fully functional, the feature extraction function has been made to be more efficient. Before this fix the algorithm would just output constant 1s after training. Final code is provided in Appendix B. After fixing up this issue all of the previous has been repeated, however the SVM stays inaccurate still.

b) Test video sequence 9995

After thoroughly testing video sequence 9996 another video has been annotated to find out how the algorithm behaves. It is the 9995 video sequence. The annotated region of interest is as follows:

- x=129"
- y=212
- height=89
- width=39

Figure 11 - ROI for video 9995

Only one kiosk has been chosen to find out whether or not it works, since having multiple kiosks will only slow down the process without guarantee of the algorithm working.

One major key factor observed while training with this video, is that the algorithm and the process took significantly longer. 11800 frames are used for the training in this instance. Frames from 11801 to 16000 are used for testing. The training took about 2 hours and a half running on HPC with 20 CPUs and 8gb of memory. Testing took about 45 minutes for the first run. Following are the results of this execution:

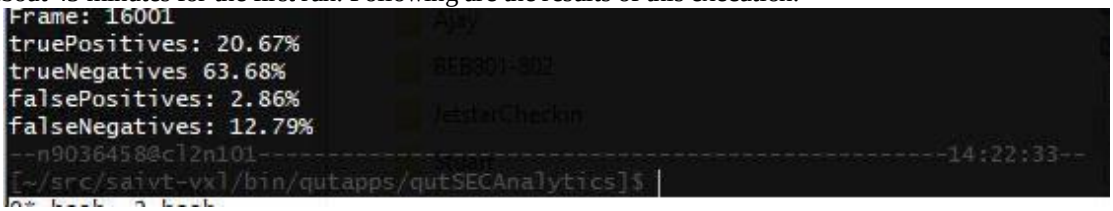

Figure 12 - Output for video 9995

As shown in figure 12 the accuracy of this algorithm is now 84.35. This suggests that the optimization did work, however there may have been issues with the previous video/setting. This however was an opportunity to push the algorithm further and to make it more efficient regardless.

Figure 13 - Mark up video 9995

Even though appropriate result was achieved, there is still room for improvement. Annotation could be made more accurate. More frames can be used for training. There is an hour worth of training and testing data, however due to the time taken for the training and testing process, less frames were used. Furthermore, inaccuracies may be caused by noise in the video as well as stuttering. The next important step for this project is to test this algorithm on multiple kiosks. Please refer to Appendix C for further information.

III. Project Management

The development method to be followed within this project is a Vee Model. This is a software development method authored by NASA and adapted in NASA's Software Management and Assurance Program. This integration methodology is to promote incremental delivery [27].

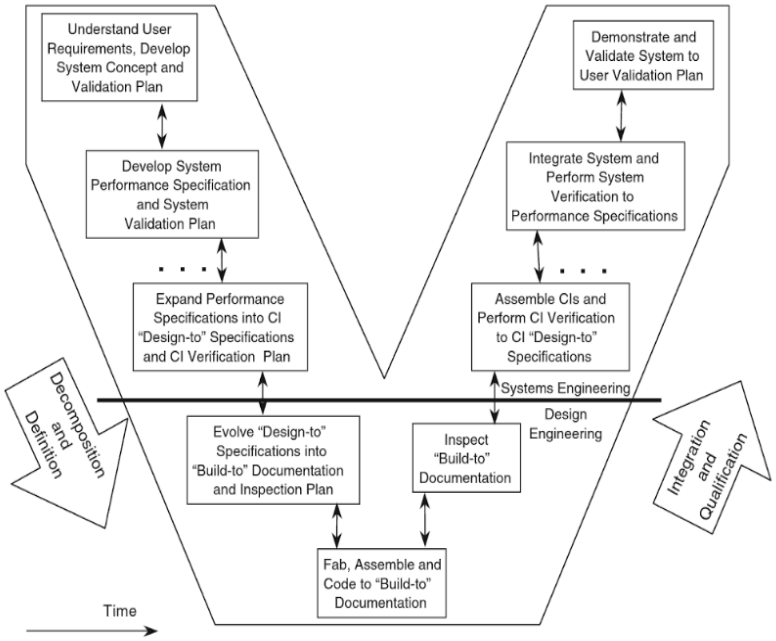

Figure 14 – Systems engineering "vee"[28, p. 10]

In the future if requirements are changed or workload does not suit the Vee model anymore, consideration is to be given to Agile/Scrum Methodologies for development. This is to have a contingency plan in case the need arises.

CONCLUSION & RECOMMENDATIONS

The prototype has been developed to detect kiosk occupancy. Such a prototype is to be further improved for deployment and implementation purposes. Various limitations of the current state of the prototype have been identified.

Overall knowledge of the systems and tools employed have been solidified through research and practical application of the methodologies discuss in the literature review.

For future reference, tools such as YOLO within DARKNET and many other algorithms and methodologies can be implemented within this and the overarching system.

The current prototype can be further implemented through further testing and implementation.

REFERENCES

[1] S. A. M. Saleh, S. A. Suandi, and H. Ibrahim, "Recent survey on crowd density estimation and counting for visual surveillance," *Eng. Appl. Artif. Intell.*, vol. 41, pp. 103–114, May 2015.

[2] D. Helbing and P. Mukerji, "Crowd disasters as systemic failures: analysis of the Love Parade disaster," *EPJ Data Sci.*, vol. 1, no. 1, p. 7, Dec. 2012.

[3] D. Ryan, S. Denman, C. Fookes, and S. Sridharan, "Scene invariant multi camera crowd counting," *Pattern Recognit. Lett.*, vol. 44, pp. 98–112, Jul. 2014.

[4] D. Ryan, S. Denman, S. Sridharan, and C. Fookes, "Scene Invariant Crowd Counting and Crowd Occupancy Analysis," in *Video Analytics for Business Intelligence*, vol. 409, C. Shan, F. Porikli, T. Xiang, and S. Gong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 161–198.

[5] R. Y. TSAI, "A efficient and accurate camera calibration technique for 3D machine vision," *Proc IEEE Conf. Comput. Vis. Pattern Recognit. CVPR 86*, pp. 364–374, 1986.

[6] R. Tsai, "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *IEEE J. Robot. Autom.*, vol. 3, no. 4, pp. 323–344, Aug. 1987.

[7] S. Denman, C. Fookes, and S. Sridharan, "Improved Simultaneous Computation of Motion Detection and Optical Flow for Object Tracking," 2009, pp. 175–182.

[8] B. Sevilla-Villanueva, K. Gibert, and M. Sànchez-Marrè, "A methodology to discover and understand complex patterns: Interpreted Integrative Multiview Clustering (I2MC)," *Pattern Recognit. Lett.*, Feb. 2017.

[9] S. Denman, V. Chandran, S. Sridharan, and C. Fookes, "A Multi-Class Tracker Using a Scalable Condensation Filter," 2006, pp. 25–25.

[10] S. Denman, C. Fookes, D. Ryan, and S. Sridharan, "Large scale monitoring of crowds and building utilisation: A new database and distributed approach," 2015, pp. 1–6.

[11] V. Lempitsky and A. Zisserman, "Learning To Count Objects in Images," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 1324–1332.

[12] Z. Ma and A. B. Chan, "Crossing the Line: Crowd Counting by Integer Programming with Local Features," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[13] B. Kim, G.-G. Lee, J.-Y. Yoon, J.-J. Kim, and W.-Y. Kim, "A Method of Counting Pedestrians in Crowded Scenes," in *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, vol. 5227, D.-S. Huang, D. C. Wunsch, D. S. Levine, and K.-H. Jo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1117–1126.

[14] D.-T. Lin and L.-W. Liu, "Real-Time Detection of Passing Objects Using Virtual Gate and Motion Vector Analysis," in *Ubiquitous Intelligence and Computing*, vol. 5061, F. E. Sandnes, Y. Zhang, C. Rong, L. T. Yang, and J. Ma, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 710–719.

[15] J. Canny, "A Computational Approach to Edge Detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.

[16] E. Rosten, R. Porter, and T. Drummond, "Faster and Better: A Machine Learning Approach to Corner Detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 1, pp. 105–119, Jan. 2010.

[17] C. E. Rasmussen, "Gaussian Processes in Machine Learning," in *Advanced Lectures on Machine Learning*, vol. 3176, O. Bousquet, U. von Luxburg, and G. Rätsch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 63–71.

[18] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: a local SVM approach," 2004, p. 32–36 Vol.3.

[19] J. K. Aggarwal and Q. Cai, "Human motion analysis: a review," 1997, pp. 90–102.

[20] J. W. Davis and A. F. Bobick, "The representation and recognition of human movement using temporal templates," 1997, pp. 928–934.

[21] V. N. Vapnik and V. Vapnik, *Statistical learning theory*, vol. 1. Wiley New York, 1998.

[22] Hao Zhang, A. C. Berg, M. Maire, and J. Malik, "SVM-KNN: Discriminative Nearest Neighbor Classification for Visual Category Recognition," 2006, vol. 2, pp. 2126–2136.

[23] Y. Lin *et al.*, "Large-scale image classification: Fast feature extraction and SVM training," in *CVPR 2011*, 2011, pp. 1689–1696.

[24] H. Meng, N. Pears, and C. Bailey, "A Human Action Recognition System for Embedded Computer Vision Application," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–6.

[25] L. Wang, *Support Vector Machines: Theory and Applications*. Springer Science & Business Media, 2005.

[26] "Extensible Markup Language (XML) 1.0 (Fifth Edition)." [Online]. Available: https://www.w3.org/TR/REC-xml/#sec-origin-goals. [Accessed: 25-May-2017].

[27] H. Mooz and K. Forsberg, "4.4.3 A Visual Explanation of Development Methods and Strategies including the Waterfall, Spiral, Vee, Vee+, and Vee++ Models," *INCOSE Int. Symp.*, vol. 11, no. 1, pp. 610–617, 2001.

[28] D. M. Buede and W. D. Miller, *The Engineering Design of Systems: Models and Methods*. John Wiley & Sons, 2016.

APPENDIX A : SIMPLE TWO IMAGE DEMO CLASSIFICATION

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/ml/ml.hpp>
#include <qututil/ImageTools.h>
#include <vul/vul_arg.h>
#include <vcl_string.h>
#include <vcl_sstream.h>
#include <vcl_iostream.h>
#include <vcl_fstream.h>


using namespace cv;



int main(int argc, char** argv)
{
    //Check input
    if( argc != 3)
    {
       std::cout <<" Usage: ImageToTrain ImageToTest" << std::endl;
       return -1;
    }

    //load image in a mat file
    Mat image;
    image = imread(argv[1], CV_LOAD_IMAGE_COLOR);
    //extract channel values and put them in a vector for training purposes
    float trainingData[271*640][3];
    int count = 0;
    for (int i = 0; i < image.rows; ++i)
    {
      for (int j = 0; j < image.cols; ++j)
      {
        for (int c = 0; c < 3 ; c++) {
          trainingData[count][c] = (float)image.at<Vec3b>(i,j)[c];
        }
        count++;
      }
    }

    //create a Matrice to train SVM
    count = 0;
    Mat trainingDataMat((271*640), 3, CV_32FC1, trainingData);

    //create the labels, this will be a one class SVM
    float labels[271*640];
    std::fill_n(labels, 271*640, 1);
    Mat labelsMat(271*640, 1, CV_32FC1, labels);

    // Set up SVM's parameters
    CvSVMParams params;
    params.svm_type    = CvSVM::ONE_CLASS;
    params.kernel_type = CvSVM::LINEAR;
    params.term_crit   = cvTermCriteria(CV_TERMCRIT_ITER, 100, 1e-6);
    params.nu = 0.6;

    // Train the SVM
    CvSVM SVM;
    SVM.train(trainingDataMat, labelsMat, Mat(), Mat(), params);

    //Load Test image in Mat file
    Mat test;
    test = imread(argv[2], CV_LOAD_IMAGE_COLOR);

    //extract channel values and put them in a vector for testing purposes
    uchar testMat[image.rows * image.cols][3];
    count = 0;
    for (int i = 0; i < image.rows; ++i)
    {
      for (int j = 0; j < image.cols; ++j)
      {
        for (int c = 0; c < 3 ; c++) {
          float intensity = image.at<Vec3b>(i,j)[c];
          testMat[count][c] = intensity;
        }
        count++;
      }
    }

    //predict
    count = 0;
    float outcome = 0;
    while(count < (image.rows * image.cols)) {
      Mat sample(1, 3, CV_32FC1, trainingData[count]);
      float result = SVM.predict(sample);
      outcome += result;
      count++;
    }

    //print outcome
    if ((outcome/count) < 0.8) {
```

```
            printf("OCCUPIED\n");
          }
          else {
            printf("FREE\n");
          }

        }
```

APPENDIX B: FINAL CODE OF THE ALGORITHM

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/ml/ml.hpp>
#include <qututil/ImageTools.h>
#include <vul/vul_arg.h>
#include <vcl_string.h>
#include <vcl_sstream.h>
#include <vcl_iostream.h>
#include <vcl_fstream.h>
#include <qutcrowd/count/qutcrowd_count.h>
#include <qutopencv/qut_video_read.h>
#include <qutopencv/qut_video_write.h>
#include <vil/algo/vil_orientations.h>
#include <qutcfg/XMLFileDataSource.h>
#include <vil/vil_crop.h>
#include <vil/vil_save.h>
#include <qutapps/qutOpAnalytics/CrowdUtils.h>

#define HISTOGRAM_BINS 4

using namespace cv;

// ***************************************************************************
// ***************************************************************************
//  Authour: Wafi Hossain
//  This function's purpose is to train and create models for kiosk dwell time
//      by means of support vector machines. It is meant to be treated as a prototype
// ***************************************************************************
// ***************************************************************************

/*
*       This function iterates through the region of interest and
* and sums the pixel densities for the respective RegressionSetup
* hence they are arranged in a vector and returned
*/
vcl_vector<float> ExtractFeaturePerPixelCrowd(int x, int y, int height, int width,
vil_image_view<double> perPixelCrowd, int numHDivs, int numVDivs);
//Returns whether or not a number is contained within the list of events
bool FrameIsOccupied(vcl_vector<vcl_pair<int, int> > events, int frame);

/*
*       will hold data such as coordinates and ROI
*       for a single kiosk
*/
```

```
struct KioskData {
  int ROI[4];
        int id;
        vcl_vector<vcl_pair<int, int> > events;
        vcl_vector<vcl_pair<int, int> > testEvents;
};

/*
*       A few global variables that will be used to save the config data in
*/
int kiosks;
int frames;
int hDiv;
int wDiv;
int featureSize = 0;

/*
*       Metrics
*/
float falsePositives = 0;
float truePositives = 0;
float falseNegatives = 0;
float trueNegatives = 0;



int main(int argc, char** argv) {
        //
        // check for inputs    home/work/SAIVT/for_Wafi/9995-640x480.avi
home/n9036458/src/saivt-vxl/qut/qutapps/qutSECAnalytics/models/QAL/
        //
        vul_arg<vcl_string>
                arg_crowdCounterConfig("-CCcfg", "Config File",
"/home/n9036458/for_Wafi/9995.xml"),
                arg_config("-cfg", "Config File", "/home/n9036458/src/saivt-
vxl/qut/qutapps/qutSECAnalytics/configs/kioskTrainer.xml");
        vul_arg<int> arg_test("-test", "testing?", 0);
        vul_arg_parse(argc, argv);
        if(arg_config() == "") {
                vul_arg_display_usage_and_exit();
        }
        //
        // load config file and important data
        //
        XMLFileDataSource* config = new XMLFileDataSource(arg_config().c_str());
        config->load();
        frames = config->getInteger("/KioskData/frames-used/@frames");
        hDiv = config->getInteger("/KioskData/Divisions/@Hdiv");
        wDiv = config->getInteger("/KioskData/Divisions/@Wdiv");
        std::cout << "height divisions " << hDiv << " width divisions: " << wDiv <<
std::endl;
        kiosks = config->getInteger("count(/KioskData/Kiosk)");
        std::cout << "Number of kiosks: " << kiosks << std::endl;
        //
        // load the kiosks data in a vector
        //
        vcl_vector<KioskData> kd;
        for(int i = 0; i < kiosks; i++) {
```

```
                KioskData kiosk;
                kiosk.id = i+1;
                vcl_stringstream currentKiosk;
                currentKiosk << "/KioskData/Kiosk[" << (i + 1) << "]";
                kiosk.ROI[0] = config->getInteger(currentKiosk.str() +
"/ROI/point/@x");
                kiosk.ROI[1] = config->getInteger(currentKiosk.str() +
"/ROI/point/@y");
                kiosk.ROI[2] = config->getInteger(currentKiosk.str() +
"/ROI/dimensions/@height");
                kiosk.ROI[3] = config->getInteger(currentKiosk.str() +
"/ROI/dimensions/@width");
                int numEvents = config->getInteger("count(" + currentKiosk.str() +
"/Event)");
                std::cout << "Kiosk: " << i+1 << " x: " << kiosk.ROI[0] << std::endl;
                std::cout << "y: " << kiosk.ROI[1] << " height: " << kiosk.ROI[2] << "
width: " << kiosk.ROI[3] << std::endl;
                //
                // load events for training
                //
                for (int c = 0; c < numEvents; c++) {
                        vcl_pair<int, int> pair;
                        vcl_stringstream currentEvent;
                        currentEvent << currentKiosk.str() << "/Event[" << c + 1 << "]";
                        pair = vcl_make_pair(config->getInteger(currentEvent.str() +
"/@start"), config->getInteger(currentEvent.str() + "/@end"));
                        kiosk.events.push_back(pair);
                }
                //
                // load the events for testing
                //
                numEvents = config->getInteger("count(" + currentKiosk.str() +
"/TestEvent)");
                for (int c = 0; c < numEvents; c++) {
                        vcl_pair<int, int> pair;
                        vcl_stringstream currentEvent;
                        currentEvent << currentKiosk.str() << "/TestEvent[" << c + 1 <<
"]";
                        pair = vcl_make_pair(config->getInteger(currentEvent.str() +
"/@start"), config->getInteger(currentEvent.str() + "/@end"));
                        kiosk.testEvents.push_back(pair);
                }
                kd.push_back(kiosk);
        }
        //
        // load crowd counter
        //
        XMLFileDataSource* crowdCounterConfig = new
XMLFileDataSource(arg_crowdCounterConfig().c_str());
        crowdCounterConfig->load();
        qutcrowd_count* cc;
        cc = new qutcrowd_count();
        cc->TrainExtractedFeatures(crowdCounterConfig);
        std::cout << "Crowd Counter is now loaded" << std::endl;
        //
        // Save this variable to see what mode this is to be ran
        //
        int testing = arg_test();
```

```
        //
        // execute the training routine with the sample data
        //
        if(testing == 0) {
                std::cout << "Will try to create feature arrays" << std::endl;
                //
                // prepare arrays for SVM
                //
                Mat labelsMat((kiosks*frames),1, CV_32FC1);
                std::cout << "feature arrays have been created" << std::endl;
                //
                // load videos
                //
                qut_video_read<unsigned char>* normalVid =  new qut_video_read<unsigned
char>(config, "/KioskData/qut-video-normal");
                qut_video_read<unsigned char>* motionVid =  new qut_video_read<unsigned
char>(config, "/KioskData/qut-video-motion");
                int countLabel = 0;
                //
                // create vector to hold features that will be then converted to
                // array for openCV integration
                //
                vcl_vector<vcl_vector<float> > featuresVector;
                std::cout << "Starting to extract feature" << std::endl;
                //
                // go through frames and extract features
                //
                while((normalVid->Next() && (normalVid->Frame() <= frames)) &&
(motionVid->Next() && (motionVid->Frame() <= frames))) {
                        std::cout << "Processing Frame: " << (normalVid->Frame()+1) <<
std::endl;
                        vil_image_view<unsigned char> im = *normalVid->Current();
                        vil_image_view<unsigned char> motion = *motionVid->Current();
                        cc->Process(im, &motion);
                        vil_image_view<unsigned int> blobID = cc->GetBlobIDMap();
                        vil_image_view<float> densityMap = cc->GetDensityMap();
                        vil_image_view<double> perPixelCrowd;
                        PerPixelCrowdDensity(blobID, densityMap,
cc->GetBlobEstimates()[0], perPixelCrowd);
                        for(int k = 0; k < kiosks; k++) {
                                if(FrameIsOccupied(kd[k].events,
(normalVid->Frame()+1))) {
                                        labelsMat.at<float>(countLabel, 0) = 1.0;
                                        std::cout << "occupied: " << 1.0 << std::endl;
                                        std::cout << "labelsMat: " <<
labelsMat.at<float>(countLabel, 0) << std::endl;
                                }
                                else {
                                        labelsMat.at<float>(countLabel, 0) = -1.0;
                                        std::cout << "occupied: " << -1.0 << std::endl;
                                        std::cout << "labelsMat: " <<
labelsMat.at<float>(countLabel, 0) << std::endl;
                                }
                                vcl_vector<float> temp =
ExtractFeaturePerPixelCrowd(kd[k].ROI[0], kd[k].ROI[1], kd[k].ROI[2], kd[k].ROI[3],
perPixelCrowd, hDiv, wDiv);
                                if(featureSize == 0) {
                                        featureSize = (int)temp.size();
```

```
                        }
                        featuresVector.push_back(temp);
                        countLabel++;
                }
        }
        //
        // convert FeatureVector to Mat for integration
        //
        Mat trainingDataMat((kiosks*frames), featureSize, CV_32FC1);
        std::cout << "Converting vector to Mat" << std::endl;
        for(int i = 0; i < kiosks*frames; i++) {
                for(int k = 0; k < featureSize; k++) {
                        trainingDataMat.at<float>(i, k) = featuresVector[i][k];
                }
        }




        //
        // Set up SVM's parameters
        //
        CvSVMParams params;
        params.svm_type    = CvSVM::C_SVC;
        params.kernel_type = CvSVM::RBF;
        params.term_crit   = cvTermCriteria(CV_TERMCRIT_ITER, 100, 1e-6);
        params.nu = 1;
        std::cout << "Training SVM" << std::endl;
        //
        // Create and Train the SVM with parameters just created
        //
        CvSVM SVM;
        SVM.train(trainingDataMat, labelsMat, Mat(), Mat(), params);
        //
        // Save the SVM
        ///home/webmaster/saivt-vxl/bin/qutapps
        SVM.save("/home/n9036458/src/saivt-
vxl/qut/qutapps/qutSECAnalytics/models/QAL/SVM");
    } else {
        CvSVM SVM;
        SVM.load("/home/n9036458/src/saivt-
vxl/qut/qutapps/qutSECAnalytics/models/QAL/SVM");
        //
        // load videos
        //
        qut_video_read<unsigned char>* normalVid =  new qut_video_read<unsigned
char>(config, "/KioskData/qut-video-normal");
        qut_video_read<unsigned char>* motionVid =  new qut_video_read<unsigned
char>(config, "/KioskData/qut-video-motion");
        int countLabel = 0;
        int countFrame =
config->getInteger("/KioskData/TestFramesStart/@start");
        int endFrame = config->getInteger("/KioskData/TestFramesStart/@end");
        float frames = endFrame - countFrame;
        //
        // go to the designated frame
        //
        normalVid->GoTo(countFrame);
```

```
		motionVid->GoTo(countFrame);
		//
		// go through frames and test the svm
		//

		// pretty markup shit
		qut_video_write<unsigned char> outputvideo("detections.avi",
normalVid->Current()->ni(), normalVid->Current()->nj(), true);
		// end pretty markup shit

		while((normalVid->Next() && (normalVid->Frame() <= endFrame)) &&
(motionVid->Next() && (motionVid->Frame() <= endFrame))) {
			//
			// lead features at each frame
			//
			vil_image_view<unsigned char> im = *normalVid->Current();
			vil_image_view<unsigned char> motion = *motionVid->Current();
			cc->Process(im, &motion);
			vil_image_view<unsigned int> blobID = cc->GetBlobIDMap();
			vil_image_view<float> densityMap = cc->GetDensityMap();
			vil_image_view<double> perPixelCrowd;

			// some pretty markup stuff
			vil_cv_image_view<unsigned char> markup;
			markup.deep_copy(im);
			// end pretty markup stuff

			PerPixelCrowdDensity(blobID, densityMap,
cc->GetBlobEstimates()[0], perPixelCrowd);
			std::cout << "got estimate"<< std::endl;
			for(int k = 0; k < kiosks; k++) {
				std::cout << "extracting feature "<< std::endl;
				vcl_vector<float> temp =
ExtractFeaturePerPixelCrowd(kd[k].ROI[0], kd[k].ROI[1], kd[k].ROI[2], kd[k].ROI[3],
perPixelCrowd, hDiv, wDiv);

				//testData = new float[(int)temp.size()];

				Mat sample(1, (int)temp.size(), CV_32FC1);
				for(int j = 0; j < (int)temp.size(); j++) {
					sample.at<float>(0, j) = temp[j];
				}
				std::cout << "Predicting "<< std::endl;
				float result = SVM.predict(sample);
				std::cout << "Kisok: " << k+1 << " Prediction: " <<
result << std::endl;
				//
				// Calculate Metrics
				//
				if(FrameIsOccupied(kd[k].testEvents,
(normalVid->Frame()+1)) && result == 1.0) {
					truePositives++;
				}
				else if(FrameIsOccupied(kd[k].testEvents,
(normalVid->Frame()+1)) && result == -1.0){
					falseNegatives++;
				}
```

```
                                else if(!FrameIsOccupied(kd[k].testEvents,
(normalVid->Frame()+1)) && result == -1.0) {
                                        trueNegatives++;
                                }
                                else if(!FrameIsOccupied(kd[k].testEvents,
(normalVid->Frame()+1)) && result == 1.0) {
                                        falsePositives++;
                                }
                                // pretty markup shit
                                if (result > 0)
                                {
                                        for (int x = kd[k].ROI[0]; x < (kd[k].ROI[0] +
kd[k].ROI[3]); x++)
                                        {
                                                for (int y = kd[k].ROI[1]; y <
(kd[k].ROI[1] + kd[k].ROI[2]); y++)
                                                {
                                                        markup(x, y, 1) =
vcl_min(markup(x, y, 1) + 50, 255);
                                                }
                                        }
                                }
                                // end pretty markup shit
                        }
                        std::cout << "Frame: " << normalVid->Frame()+1 << std::endl;

                        // pretty markup shit
                        outputvideo.Write(markup);
                        // end pretty markup shit
                }

                std::cout << std::fixed;
                std::cout << std::setprecision(2);
                std::cout << "truePositives: " << (truePositives/(frames*kiosks))*100
<< "%" << std::endl;
                std::cout << "trueNegatives " << (trueNegatives/(frames*kiosks))*100 <<
"%" << std::endl;
                std::cout << "falsePositives: " << (falsePositives/(frames*kiosks))*100
<< "%" << std::endl;
                std::cout << "falseNegatives: " << (falseNegatives/(frames*kiosks))*100
<< "%" << std::endl;
                // pretty markup shit
                outputvideo.Close();
                // end pretty markup shit
        }
}

vcl_vector<float> ExtractFeaturePerPixelCrowd(int x, int y, int height, int width,
vil_image_view<double> perPixelCrowd, int numHDivs, int numVDivs) {
        std::cout << "height: " << height << " width: " << width << std::endl;
        double h = double(height)/double(numHDivs);
        double w = double(width)/double(numVDivs);

        std::cout << "h: " << h << " w: " << w << std::endl;
        vcl_vector<float> features;
        vil_image_view<double> image_ROI = vil_crop(perPixelCrowd, x, width, y,
height);
        vil_image_view<unsigned char> im_grey;
```

```
        vil_convert_cast(image_ROI, im_grey);

        vil_image_view<unsigned char> orient_im;

        vil_image_view<float> grad_mag;
        vil_orientations_from_sobel(im_grey, orient_im, grad_mag, 2 * 8);
        for(double i = 0; i < image_ROI.ni()-1; i +=w) {
                for(double j = 0; j < image_ROI.nj()-1; j += h) {
                        float value = 0;
                        for(double ii = i; (ii < (i + w)) && (ii < image_ROI.ni());
ii++) {
                                for(double jj = j; (jj < (j + h)) && (jj <
image_ROI.nj()); jj++) {
                                        value += (float)image_ROI(ii,jj);

                                }
                        }
                        features.push_back(value);
                        double histvalues[8];
                        histvalues[0] = 0; histvalues[1] = 0; histvalues[2] = 0;
histvalues[3] = 0; histvalues[4] = 0; histvalues[5] = 0; histvalues[6] = 0;
histvalues[7] = 0;
                        for(double ii = i; ii < (i + w) && ii < image_ROI.ni(); ii++) {
                                for(double jj = j; jj < (j + h) && jj < image_ROI.nj();
jj++) {
                                        histvalues[orient_im(ii, jj) % 8] += 1;
                                }
                        }
                        double total = histvalues[0] + histvalues[1] + histvalues[2] +
histvalues[3] + histvalues[4] + histvalues[5] + histvalues[6] + histvalues[7];


                        features.push_back(histvalues[0]/total);
                        features.push_back(histvalues[1]/total);
                        features.push_back(histvalues[2]/total);
                        features.push_back(histvalues[3]/total);
                        features.push_back(histvalues[4]/total);
                        features.push_back(histvalues[5]/total);
                        features.push_back(histvalues[6]/total);
                        features.push_back(histvalues[7]/total);
                }
        }

        return features;
}

bool FrameIsOccupied(vcl_vector<vcl_pair<int, int> > events, int frame) {
        for(int i = 0; i < events.size(); i++) {
                if(frame >= events[i].first && frame <= events[i].second)
                        return true;
        }
        return false;
}
```

APENDIX C: CONFIG FILE FOR WORKING CONFIGURATION

```
<KioskData>
	<qut-video-normal path="/home/n9036458/for_Wafi/9995-640x480.avi" />
	<qut-video-motion path="/home/n9036458/for_Wafi/9995-640x480-motion.avi" />
	<TestFramesStart start="11801" end="16000" />
	<frames-used frames="11800" />
	<Divisions Hdiv="4" Wdiv="4"/>
	<Kiosk>
		<ROI>
		<point x="129" y="212"/>
		<dimensions height="89" width="39"/>
		</ROI>
		<Event start="0" end="435" />
		<Event start="460" end="3482" />
		<Event start="3493" end="3511" />
		<Event start="3707" end="3722" />
		<Event start="4057" end="3066" />
		<Event start="4120" end="4132" />
		<Event start="4308" end="4132" />
		<Event start="4326" end="11307" />
		<Event start="11389" end="11403" />
		<Event start="11532" end="11542" />
		<Event start="11594" end="11603" />
		<Event start="11631" end="11654" />


		<TestEvent start="11804" end="13044" />
		<TestEvent start="13390" end="13401" />
		<TestEvent start="13587" end="13605" />
		<TestEvent start="14016" end="14031" />
		<TestEvent start="14429" end="14442" />
		<TestEvent start="14629" end="14637" />
		<TestEvent start="14804" end="14812" />
		<TestEvent start="14891" end="14927" />
		<TestEvent start="15243" end="15258" />
		<TestEvent start="15369" end="15382" />
		<TestEvent start="15769" end="15786" />


	</Kiosk>
</KioskData>
```

APENDIX D: GANTT CHART FOR SEMESTER 1