PEDESTRIAN DETECTION AND TRACKING IN SURVEILLANCE VIDEO

By

PENNY CHONG

A project report submitted in partial fulfilment of the
requirements for the award of Bachelor of Science (Hons.)
Applied Mathematics with Computing

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2016

DECLARATION OF ORIGINALITY

I hereby declare that this project report entitled "**PEDESTRIAN DETECTION AND TRACKING IN SURVEILLANCE VIDEO**" is my own work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : _____

ID No. : _____

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"PEDESTRIAN DETECTION AND TRACKING IN SURVEILLANCE VIDEO"** was prepared by **PENNY CHONG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Science (Hons.) Applied Mathematics with Computing at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : _____

Date : _____

ACKNOWLEDGEMENTS

I would like to express my utmost gratitude to Universiti Tunku Abdul Rahman for providing an opportunity for me to conduct this research as a partial fulfilment of the requirement for the degree in Bachelor of Science (Hons) Applied Mathematics with Computing.

Throughout the development of this research, I am very fortunate to be blessed with advice and guidance from my supervisor, Dr. Tay Yong Haur. Without his help, I would not be able to complete this research project.

In addition, I would also like to thank my loving parents and friends who had motivated and encouraged me for this one year.

PENNY CHONG

PEDESTRIAN DETECTION AND TRACKING IN SURVEILLANCE VIDEO

PENNY CHONG

ABSTRACT

Pedestrian detection and tracking has many important applications in the security industry, pedestrian demographic analysis, and intelligent transportation system (ITS). In this project, we will develop a stable pedestrian detection and tracking algorithm. The Town Centre video frames and the hand annotated ground truth published by the University of Oxford are used as a benchmark. In experiment 1, we used Dalal and Triggs (2005) Support Vector Machines (SVM) classifier to detect pedestrians. In experiment 2, we trained our own cascade of boosted classifiers with Histogram of Oriented Gradients (HOG) feature for detection. Using Daimler training samples and INRIA training samples, we have trained two different detectors to perform pedestrian detection. The detector trained with Daimler training samples has outperformed the detector trained with INRIA training samples. For both experiments, the raw detection results are passed to the tracker. Our tracker uses Kalman filter to estimate the location of the pedestrians based on their track history. For data association, we employed the Hungarian algorithm. Overall, experiment 2 shows a more promising result as compared to experiment 1. Using raw detections from Daimler detector, our multiple object tracking accuracy (MOTA) value in experiment 2 had surpassed Benfold and Reid (2011) MOTA value by approximately 1%. However, it was observed that our algorithm suffers from a high number of misses due to occlusion. This is a common problem especially in crowded or semi-crowded environment. Thus to improve the detection or tracking results, one can opt to use a part-based detector instead of a full body detector to estimate the location of the pedestrians.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER 1: INTRODUCTION

1-1 Background

Computer or machine vision is a field in artificial intelligence that has become a great area of research due to its wide range of applications. From automated navigation of vehicles to medical imaging, all these applications require the computer to process, understand and analyse a scenario in order to make intelligent decisions. Putting it in layman terms, computer vision simply means teaching the computer to mimic the human eyes and brain.

With the power of computers today and the current breakthrough in technologies, there are now various methods/algorithms that were developed to enable a computer/machine to perform tasks such as object detection, object tracking and pattern recognition. In this study, the focus will be on object detection and tracking with pedestrian as our object of interest.

Pedestrian detection and tracking, have important roles in the security industry, pedestrian demographic analysis, and intelligent transportation system (ITS). In the security industry, an automated human detection and tracking system is necessary as an increasing number of surveillance cameras /CCTVs are installed each day. This intelligent surveillance system facilitates the analysis of video footages. Therefore, it has now become impractical for human operators to monitor the massive video footages like before.

Besides the security industry, pedestrian detection and tracking also play a major role in pedestrian demographic systems especially in analyzing the demographic distribution of a crowd. Moreover in an overcrowding situation, it serves as a counting system.

Likewise, pedestrian detection and tracking can also be implemented in the braking system of cars as a safety feature. In the event where the driver is about to hit a pedestrian, the car will automatically brake on its own. With this implementation, the chances of a car hitting a pedestrian is reduced.

1-2 Aims and Objectives

A study is done on how pedestrian detection and tracking can be applied and used in real-word applications. For pedestrian detection, we will use Histogram of Oriented Gradients (HOG) feature descriptors as a feature representation of the human shape. For tracking, we will use Kalman filter and the detection results to predict the path of the pedestrian. At the end of this project, we will deliver a stable pedestrian detection and tracking algorithm.

1-3 Problem Statement

The development of a reliable algorithm to perform pedestrian detection and tracking is still a challenge today. Many had underestimated the complexity of this problem as detection and tracking is a process that can be easily done by the human eyes. However for a computer to model and imitate the human eyes, there are many challenges involved. One of the challenges faced in pedestrian detection and tracking is the variation of heights and body shapes of pedestrians.

Like any other fixed objects, pedestrians may come in different/same heights and body shapes. These features may or may not be helpful in the tracking process. In an extreme case where two pedestrians have the same height, body shape and are wearing similar outfits, it may be quite difficult for a computer/machine to differentiate and distinguish their paths. Therefore having the same height and body shape is not helpful in this case.

Another challenge faced in the detection and tracking process is occlusion. There are many types of occlusion that can occur in a real-time scenario. Occlusion between pedestrians, occlusion between pedestrians and buildings, occlusion between pedestrians and vehicles are the common types of occlusions faced in a real-time scenario. In an overcrowded situation, all these occlusions may affect the accuracy of the algorithm.

In addition, the problem becomes more complex due to illumination changes in the scene. Different lighting conditions may affect the visibility of an object and even alter the appearance of the object. Hence the way lights are placed in a scene, may cause an object to look different. In our case, pedestrians may look different due to the lighting conditions in the scene.

Therefore it is not easy for one to develop a reliable real-time pedestrian detection and tracking algorithm that is able to address all these issues. Although many other researchers have employed different approaches to address this particular problem, there is still no promising algorithm in terms of accuracy and speed. Hence, a study is carried out to understand and address these issues.

1-4 Project Scope

In this project, our attention will be on developing an algorithm to detect and track multiple pedestrians in surveillance videos. This project is only limited to semi-crowded environment in a stationary single camera view.

For a fair comparison, the standard Town Centre video and the hand annotated ground truth data provided by the University of Oxford were used. To compare our results with other researchers, we benchmark our final results against theirs using the same evaluation algorithm.

CHAPTER 2: LITERATURE REVIEW

2-1 Classification of Object Detection Methods

Generally there are three different types of approach used in object detection. They are the feature-based approach, motion-based approach and classifier-based approach (Shantaiya, Verma and Mehta, 2013). In feature-based approach, the features in the image are extracted as a representation of the image. Detection using colors and shapes are examples of a feature-based detection. In motion-based approach, the objects are detected based on movements. Hence, only moving objects can be detected while stationary objects cannot be detected. Detection using background subtraction is an example of a motion-based approach. In a classifier-based approach, the classifier is trained to recognize or detect objects by feeding in positive and negative training samples. Examples of classifier-based approach are Support Vector Machine (SVM) classifier and cascade of boosted classifiers.

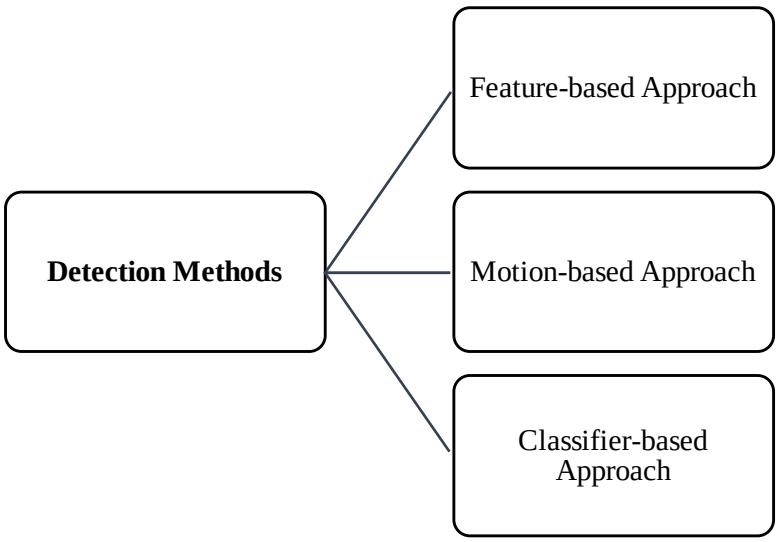

Figure 2.1: Classification of detection methods

2-1-1 Feature-based Approach

Tian and his team of researchers had employed a Histogram of Oriented Gradients (HOG) feature descriptor and a color feature in pedestrian detection (Tian, et al., 2013). HOG with Local Color Self Similarity Feature (LCSSF) instead of Color Self Similarity (CSS) had speed up detection time and performance. From this feature extraction process, one can obtain useful information on the contours and the distribution of colors.

2-1-2 Motion-based Approach

Rakibe and Patil (2013) had employed background subtraction to detect humans in video frames. The moving humans were detected by finding the difference between the current frame and the background. Thus the background needs to be updated from time to time so that the algorithm is more robust to illumination changes in the scene. However, this approach will detect all moving objects including the objects that are not of our interest.

2-1-3 Classifier-based Approach

Dalal and Triggs (2005) had used a support vector machine (SVM) binary classifier with HOG features to detect humans. Support vector machine will find an optimal hyperplane that splits the training samples into groups. With the optimal hyperplane, the SVM classifier can group the objects according to their class.

On the other hand, Viola and Jones (2001) had proposed to use a cascade of boosted classifiers for face detection. The cascade was designed in such a way that it speeds up the detection process. With several layers in the cascade, a large number of negative sub-windows can be eliminated quickly which speeds up the face detection process.

2-2 Classifications of Object Tracking Methods

Generally there are three different types of approach used in object tracking. Point tracking approach, kernel-based tracking approach and silhouette-based tracking approach (Athanesious and Suresh, 2012). In point tracking approach, we represent the detected objects as points across frames. This approach is capable of tracking very small objects. The examples of a point tracking approach are Kalman filter, particle filter and Multiple Hypothesis Tracking (MHT). In kernel tracking, a moving object is computed and represented by an embryonic object region from frame-to-frame. The motion will be represented in the form of a parametric function such as conformal, translation and affine (Yilmaz, et al., 2006). The examples of a kernel tracking are Simple Template Matching, Kanade-Lucas-Tomasi (KLT) and Mean Shift Method. For a silhouette-based tracking, we use colour histogram, contour or edges to model the objects. The examples of silhouette tracking are contour tracking and shape matching. Despite the three different approaches that can be used in object tracking, many researchers have employed a point tracking approach in pedestrian tracking.

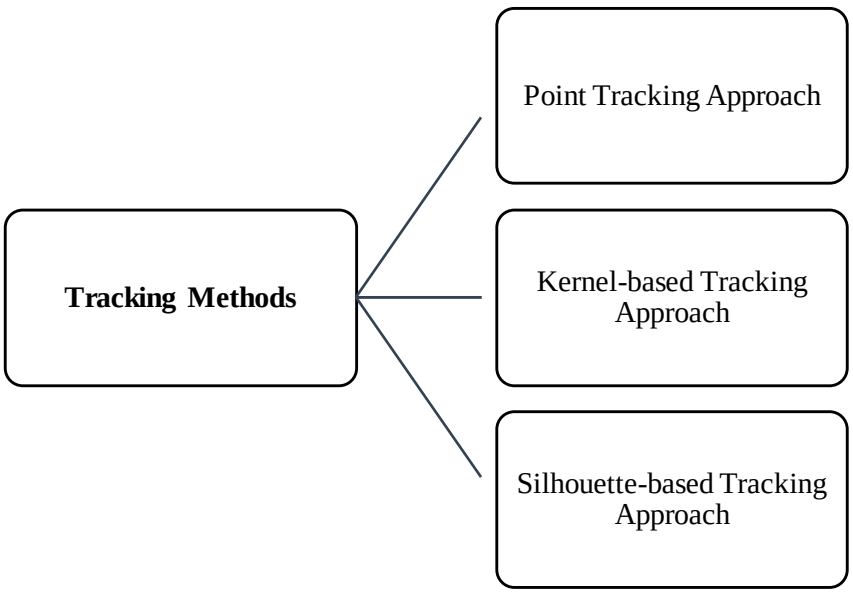

Figure 2.2: Classification of tracking methods

2-2-1 Point Tracking Approach

Jiang and his team of researchers had used a point tracking approach. They had employed a colour model together with a Kalman filter motion model in the tracking of pedestrian (Jiang, et al., 2010). Initially, a Histogram of Oriented Gradient (HOG) and a classification with a linear Support Vector Machine (SVM) as proposed by Dalal and Triggs (2005) were used to detect the pedestrians. After the detection phase, a 4-dimensional colour histogram for each detected pedestrian window were extracted out and compared using the Bhattacharyya coefficient. In addition to the 4-D color histogram, a prediction motion model called the Kalman filter was also used to predict the path/trajectory of each pedestrian. The tracking results obtained using a colour model and motion model had outperformed the results obtained using a colour model only.

Similarly, a point tracking approach using an infrared sensor supplied with information from the road network was employed in the tracking of pedestrian (Skoglar, et al., 2012). Road network information was send to a multiple model particle filter to enhance tracking performance. This multiple model consists of an on-road (road-constrained) model and an off-road (road-unconstrained) model that is used to track targets in environments similar to parks.

Likewise an approach involving sparse infrastructure support with particle filter was employed by Jin, Soh, Motani and Wong (2013) to solve indoor pedestrian tracking. They had considered a Dead Reckoning (DR) and a ranging sub-system with a sparse infrastructure to be used. To bound the error in tracking, a particle filter is applied by fusing DR with sparse range measurements.

2-2-2 Kernel-based Tracking Approach

Benfold and Reid (2011) had employed a kernel approach in tackling the tracking problem. They had proposed a stable multi-target tracking algorithm using Kanade-Lucas-Tomasi (KLT) to track pedestrians in a real-time surveillance video. In order to achieve a time-efficient algorithm, they had employed a multi-threaded approach where one thread is responsible for an asynchronous Histogram of Oriented Gradient (HOG) detection, a second thread will perform a KLT feature point tracking task, a third thread will carry out the data association task using Markov-Chain Monte-Carlo Data Association (MCMCDA)

technique and finally a fourth thread will generate and optimise the output. The system was evaluated using the standard CLEAR MOT evaluation criteria and was found to be capable of giving a precise estimate on the location of pedestrians in a large crowd.

2-2-3 Silhouette-based Tracking Approach

For objects with complex shapes and cannot be represented by a set of points, silhouette based tracking approach is more appropriate. These complex objects are represented by a silhouette for a better shape description. For instance, Sato and Aggarwal had employed a silhouette matching technique using a Hough transform to calculate the trajectory of the moving object (Sato and Aggarwal 2004, cited in Han, et al., 2009).

CHAPTER 3: RESEARCH METHODOLOGY

3-1 System Overview

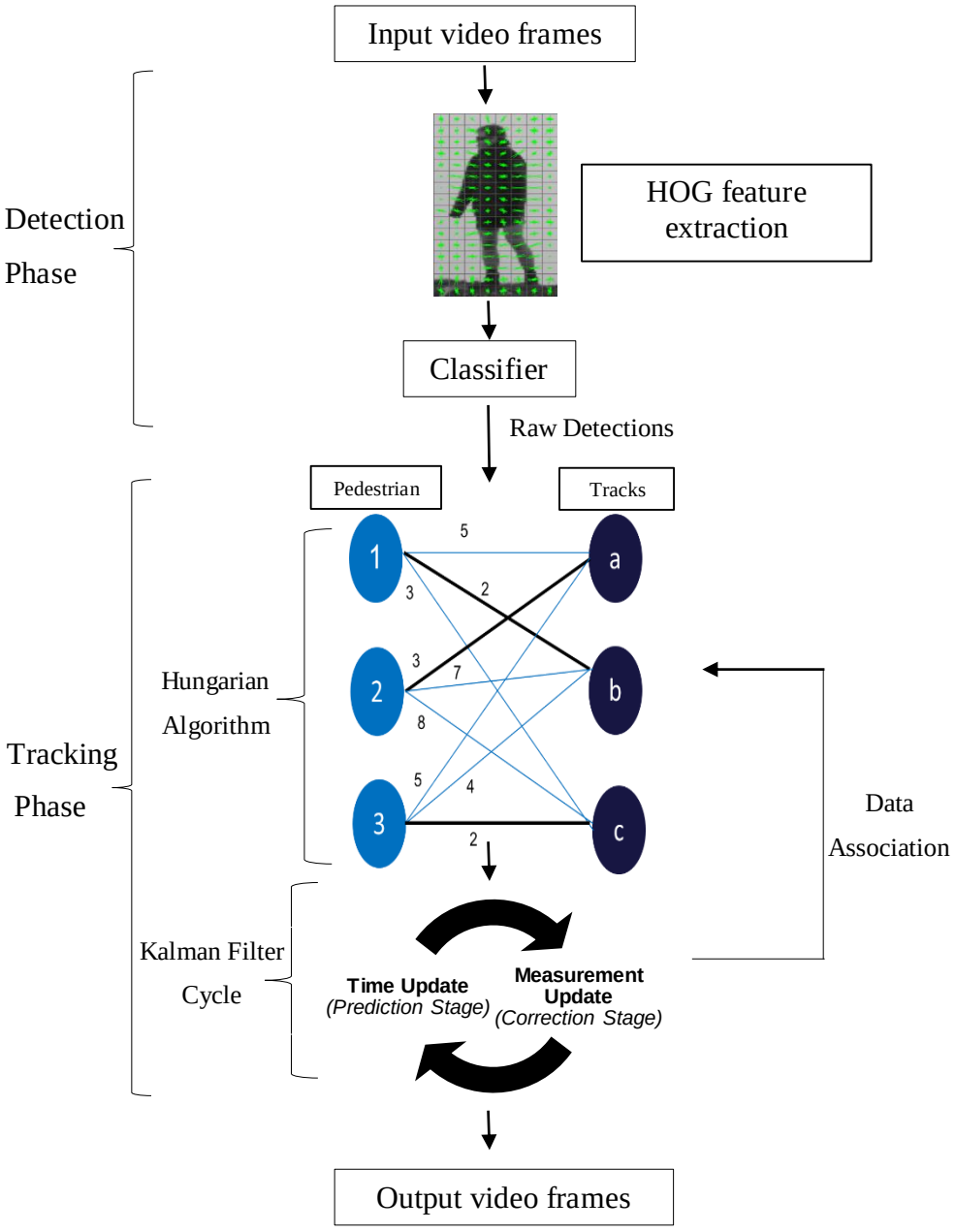

Figure 3.1: System overview of detection and tracking process

Generally, the algorithm is divided into two phases. The first phase is the detection phase. After the detection phase, is the tracking phase. The Town Centre data set (video) provided by the University of Oxford is used to test the performance of our algorithm. It is a 3 minutes full HD video with 25 frames per second and a resolution of 1920 x 1080. The video has an average of 16 pedestrians per frame. The algorithm is implemented using Open Source Computer Vision (OpenCV) 2.4.9 and C++ language.

3-2 Detection Phase

In the beginning, the Town Centre video is feed into the system on a frame-by-frame basis. In the feature extraction process, we compute the Histogram of Oriented Gradient feature descriptor for every image. Then the feature vectors are feed into a binary classifier, either a SVM classifier or a cascade of boosted classifiers. Then the raw detection results from the classifier are used in the tracking phase.

3-3 Tracking Phase

Each detected pedestrian is assigned a unique identity. We employ the Hungarian algorithm on the raw detection results to find correspondences between the detections and existing tracks. Then, we use Kalman filter to estimate the current location of each pedestrian based on his/her previous track history. Next in the measurement update stage of the Kalman filter cycle, the filter uses information from the raw detections, to correct and refine its prediction. Again, we employ the Hungarian algorithm to find correspondences between the predictions and the tracks. Upon finding those correspondences, the filter continue to predict the location of the pedestrian using the track history and raw detection results. The tracking results are stored in json format and will be used to evaluate the tracking performance.

3-4 Evaluation Method

Two different type of performance measures are used. For detection, we use a receiver operating characteristic (ROC) curve to study the trade-off between hit rate and false positive rate during detection. On the other hand, we will use the CLEAR MOT Metrics to measure our tracking results based on Multiple Object Tracking Precision (MOTP) and Multiple Object Tracking Accuracy (MOTA) values. For both methods, we will use the ground truth data provided by the University of Oxford and compare them against our results.

3-4-1 Evaluation for Detection

The ROC curve is a plot that enables us to study the trade-off between the true positive rate (hit rate) and the false positive rate. The ROC plot is also known as a plot of sensitivity against 1-specificity. A good classifier will have a high hit rate with a low false positive rate. Hence the classifier whose curve is the closest to the top left corner will have the best discriminative ability or in other words, a better detection performance. However if a classifier's curve drops below the reference line (y=x) as indicated in the figure below, then the classifier is said to perform far worse than a random classifier (Choi, n.d.).

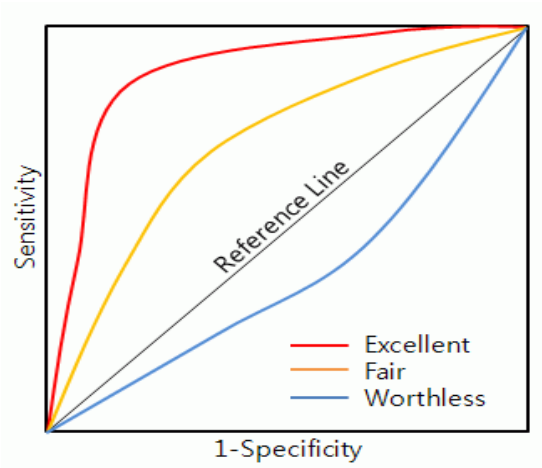

Figure 3.2: Comparison of ROC curves

3-4-2 Evaluation for Tracking

To evaluate our tracking performance, we calculate the MOTP and MOTA values. For MOTP, the errors representing the differences in the object's true and estimated positions are computed by comparing each correspondence found. On the other hand, configuration errors such as misses *(the number of objects in ground truth that are not in the results),* false positives *(tracker results for which no ground truth exists)* and mismatches *(event where the tracker results changed as compared to past frames such as the swap in identities)* are accounted for in the calculation of MOTA (Bernadin and Stiefelhagen, 2008). Higher values for MOTP and MOTA indicate better tracking performance.

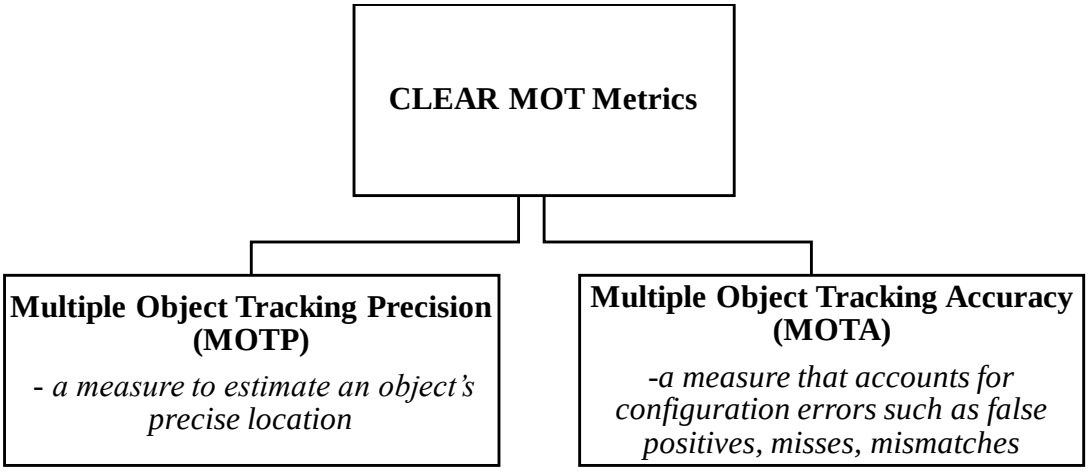

Figure 3.3: Tracking evaluation – CLEAR MOT metrics

CHAPTER 4: DETECTION ALGORITHM

4-1 Histogram of Oriented Gradients (HOG)

Human detection in a crowded environment is a challenging task as human comes in various shapes and heights. Human can also take various forms and postures. Hence for a computer to distinguish humans from the background and other objects, we need a robust feature to represent the human shape.

In this study, we have employed the Histogram of Oriented Gradients (HOG) feature descriptor as a feature representation for the human shape. Since human can take various form of appearances, we use intensity gradients or edge directions to represent the human shape. This method is invariant to photometric and geometric transformation (Dalal and Triggs, 2005). In other words, this feature has a certain degree of robustness towards rotation, translation and illumination changes in the environment.

Initially, ensure that the gamma values and color are normalized. Next, compute the gradient values vertically and horizontally using the one-dimensional centered, point discrete derivative mask. Based on the gradient values, the pixels cast weighted votes into orientation cells. Finally, the gradient strengths are contrast-normalized by grouping cells into large spatial connected blocks to adapt better to illumination changes.

Upon computing the HOG feature vectors, we feed them into a binary classifier. The classifier can be a support vector machine (SVM) classifier or a cascade of boosted classifiers where each layer is trained to recognize the feature vectors using the Adaptive Boosting (AdaBoost) method.

4-2 Support Vector Machine (SVM) Classifier

SVM algorithm finds an optimal hyperplane that maximizes the separation between the hyperplane and the points in space. In a binary linear problem, the SVM algorithm groups the training samples into two different categories through supervised learning. There will be many hyperplanes that can split the training samples into two distinct categories. However, only the hyperplane that maximizes the distance between the two categories is the optimal hyperplane (Osuna, Freund and Girosi, 1997). With the optimal hyperplane, the SVM classifier can classify new unseen examples into one of the category.

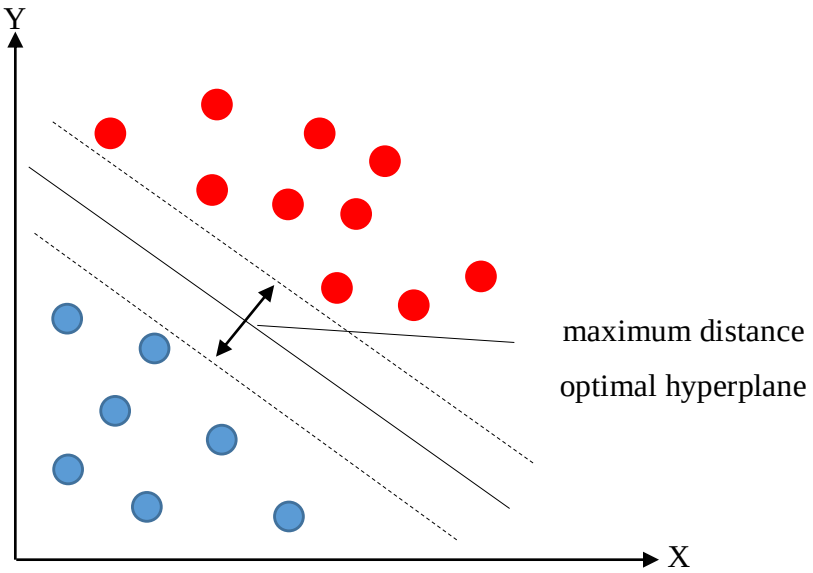

Figure 4.1: SVM classifier

In the first experiment, we will use the SVM with HOG feature pedestrian detector by Dalal and Triggs (2005) to compare against our trained cascade of boosted classifiers in our second experiment.

4-3 Cascade of Boosted Classifiers

Cascade of boosted classifiers was proposed by Viola and Jones (2001) for face detection. The cascade consists of many layers where each layer is a binary classifier trained with AdaBoost, a machine learning algorithm to classify images according to certain features. At any layer where a sub-window is rejected, there will be no further processing on that sub-window. Therefore with several layers, the computational time used to match the features is reduced. This speeds up the detection process since most of the sub-windows in a single image are negative sub-windows and can be eliminated quickly in the first few layers of the cascade. For sub-windows that are not rejected, there are passed on to the next layer where each layer is more complex than the last. Only those sub-windows that pass through all the layers are detected.

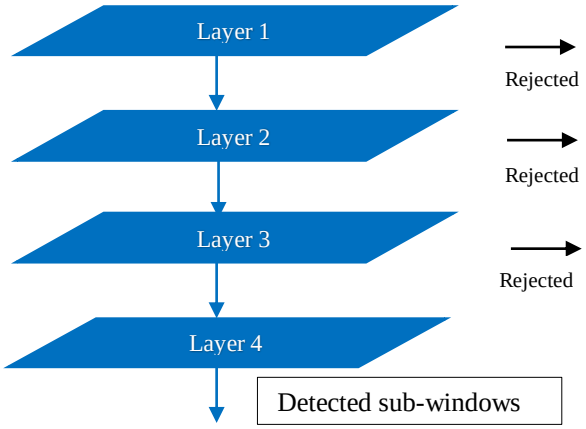

Figure 4.2: Cascade of boosted classifiers

With promising results from face detection, we follow this approach for pedestrian detection in our second experiment. However in our case, we employ HOG feature descriptor instead of Haar-like features in the feature extraction process. We have trained two different detectors using Daimler training samples and INRIA training samples. The detection results for these two detectors will be discussed in the discussion section.

CHAPTER 5: TRACKING ALGORITHM

5-1 Kalman Filter Model

The Kalman filter model is a linear quadratic estimation (LQE) model that is used to estimate the state of an object in an on-going process. The Kalman filter model is able to give an estimation of the previous, current and also the future states through recursive computations that minimizes the mean square error. These recursive computations will go through two main stages repeatedly. The two stages are the time update stage and the measurement update stage. The time update stage is also known as the prediction stage as this is a stage where the filter will predict the current state using observations from the previous state. On the other hand, the measurement update stage is also commonly known as the correction stage as this is a stage where the filter will refine its prediction and correct its estimation by taking into account the actual measurement.

In this study, a standard Kalman filter model is used to determine each pedestrian's trajectory. In using this standard Kalman filter model, the system needs to be a linear system where the state variables are normally distributed. In the case where the system is non-linear, one should use an extended Kalman filter. On the other hand, if the state variables are not normally distributed, one should use a particle filter instead of a Kalman filter.

Assuming we have a linear system where the state variables are normally distributed and a constant velocity system, the discrete Kalman filter prediction equations will have the form

$$\begin{aligned} \hat{x}_k^- &= A\hat{x}_{k-1} + Bu_{k-1} \\ P_k^- &= AP_{k-1}A^T + Q \end{aligned} \tag{5.1}$$

and the correction equations will have the form

$$\begin{aligned} K_k &= P_k^- H^T (H P_k^- H^T + R)^{-1} \\ \hat{x}_k &= \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \\ P_k &= (I - K_k H)P_k^- \end{aligned} \tag{5.2}$$

where $x \in R^n$, $z \in R^m$, $u \in R^l$

- $\hat{x}_k^-$ - prediction at time k, given observations up to time k-1
- *(priori estimate)*
- $\hat{x}_k$ - prediction at time k, given observations up to time k *(posteriori estimate)*
- P_k^- - error covariance at time k, given observations up to time k-1
- K_k - n x m Kalman gain matrix at time k
- u_{k-1} - optional control input at time k-1
- z_k - actual measurement/location
- A - n x n matrix that relates the prediction at time k-1 to the prediction at time k
- B - n x l matrix that relates the optional control input, u to the prediction x
- Q - process of noise covariance
- H - m x n measurement matrix
- R - measurements of noise covariance

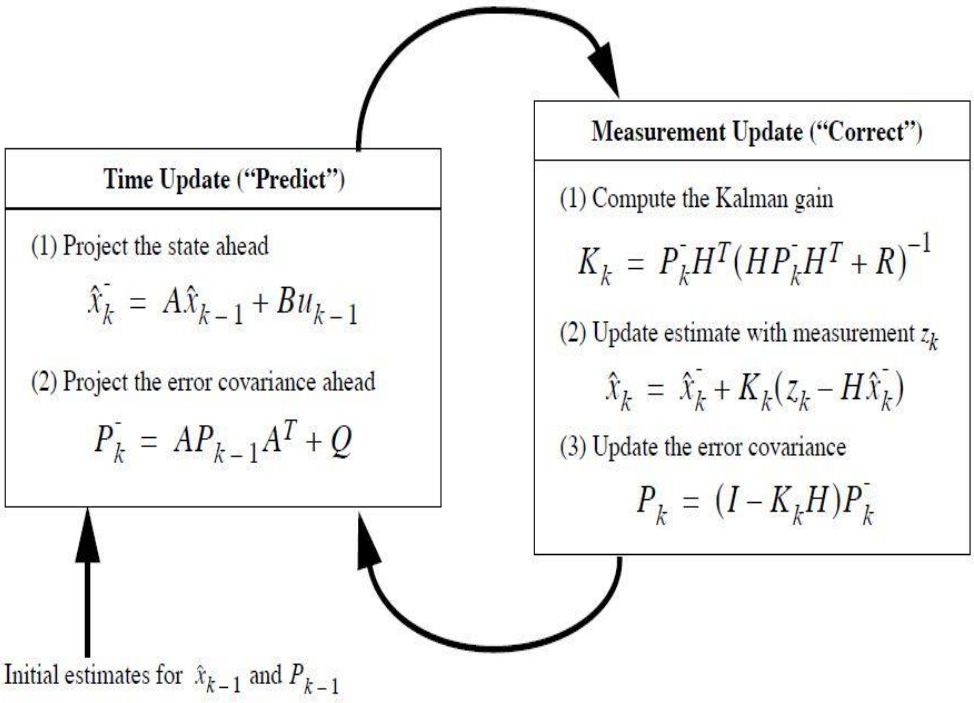

Figure 5.1: A complete picture of the Kalman filter operation (Welch and Bishop, 2006. p. 6)

In our algorithm, the Kalman filter will give an estimation of the current state (estimated centroid of the pedestrian) by revolving around two main stages, the time update stage and the measurement update stage. In the time update stage, the filter estimates the current location of the pedestrian based on his/her previous location. The output from the time update stage will be used as an input for the measurement update stage. If the pedestrian's actual location is detected and the centroid computed, the filter would use the actual location to correct its prediction in the measurement update stage. Hence this is called a filtered location/estimate.

In the case where the pedestrian's location is not detected in a particular frame, the filter will solely depend on the previous observation to predict the current location of the pedestrian. Hence the output from the measurement update stage is now used to predict the pedestrian's next location in the time update stage. In the event where the pedestrian's location is not detected for a period of time (in our case, about 10 or 15 consecutive frames), the pedestrian is assumed to have left the scene.

Continuing in this manner, the Kalman filter model works recursively to predict the current state based on the information provided by the previous state. Thus this filter is able to predict previous, current and also the future states. This self-correcting mechanism makes Kalman filter particularly useful in smoothing out noisy input and refining the pedestrian's trajectory even when there is occlusion or when the detection fails.

5-2 Data Association using Hungarian Algorithm

Multiple pedestrian tracking or more generally multiple object tracking faces more challenges as compared to a single object tracking. In multiple object tracking, the right tracks needs to be assigned to the right objects. One also needs to consider new objects entering into the scene. However the most difficult/challenging part in multiple object tracking is to maintain each object's identity and track while they merge into a single detection. Hence the problem of determining which observation is useful to a particular object of interest is known as data association.

Through data association, one can eliminate non-informative observations and associate only relevant observations to the right object. Hence, we adopted the Hungarian Algorithm to solve the data association problem in this study.

Hungarian algorithm or also known as Munkres algorithm is commonly used to solve assignments in linear programming problems. This algorithm will assign n objects to m tasks in such a way that it minimizes the cost incurred.

In this study, n x m cost matrix that represents the Euclidean distance between each detected centroid point and each pedestrian's track was constructed. The following steps were then applied to solve the data association problem (Munkres, 1957).

Step 0: The matrix is rotated in such a way that the number of columns ≥ number of rows. Then, we let k equals to the minimum of m and n.

Step 1: For each row, subtract the minimum entry of that row from each entry in that row. Move on to **Step 2**.

Step 2: Look for a zero, Z in the new matrix. Only star Z, when we cannot find a starred zero in the same column or row. Repeat for all zero elements. Move on to **Step 3**.

Step 3: Cover those columns with starred zeros. If there are k number of columns covered, then we have a whole set of unique assignments. If so, go to **DONE**, else move on to **Step 4**.

Step 4: Prime an uncovered zero. Move on to **Step 5** in the absence of a starred zero and a primed zero in the same row. Else, cover the row and uncover the column with starred zero. Repeat this so that all the zeros are covered. Among the uncovered elements, note down the smallest element and move on to **Step 6**.

Step 5: Form a series of primed and starred zeros in alternating order. Let Z_0 be an uncovered primed zero obtained from **Step 4**. If a starred zero is present in the column of Z_0, let it be Z_1. Let Z_2 be the primed zero found in the Z_1 row. Repeat and only stops at a primed zero where the column does not have a starred zero. Unstar all the starred zeros in the series. Next, star all the primed zeros in the series. Then, uncover the lines and erase all primes. Go back to **Step 3**.

Step 6: For each value obtained in **Step 4**, add to each entry in the covered rows and subtract from each entry in the uncovered columns. Go back to **Step 4.**

DONE: The location of the starred zeros in the matrix is the assignment pair. In other words if a starred zero is found in row *p* and column *q* of the cost matrix, then there is an association between the entry in row *p* and column *q*.

By applying the steps above, we are able to assign the correct track to each pedestrian.

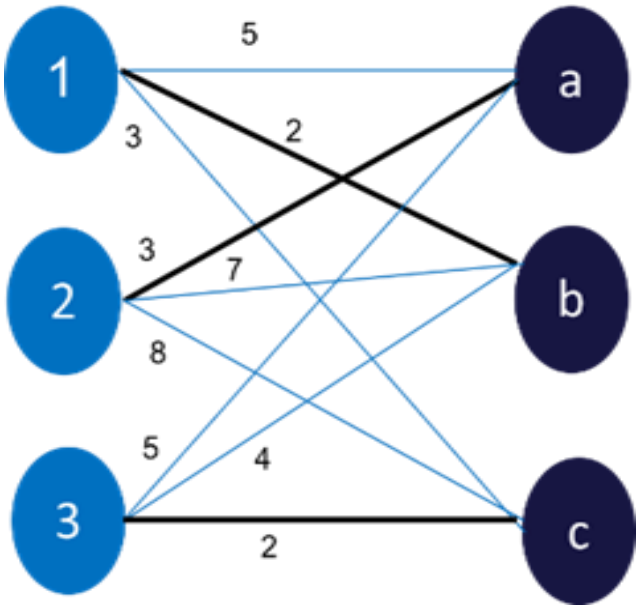

Figure 5.2: Hungarian algorithm

CHAPTER 6: EXPERIMENTAL SETUP

6-1 Experiment 1 Setup

In the first experiment, we input the Town Centre video frames into our system and select appropriate detection and tracking parameters. We used the OpenCV default people detector to perform pedestrian detection. This default people detector is an implementation of the Dalal and Triggs (2005) human detector. We do not train our own SVM classifier to perform pedestrian detection in this experiment.

The raw detections obtained from the default detector in OpenCV library are passed on to our tracker. Our tracker predicts the paths of the pedestrian with Kalman filter based on the raw detections. At the end of the 3 minutes video, the tracking ouput which contains information on the location of the pedestrians will be stored in a json format. This json file together with the ground truth will be used to evaluate our tracking performance.

6-2 Experiment 2 Setup

In the second experiment, we trained our own cascade of boosted classifiers instead of using the default people detector in OpenCV library to perform pedestrian detection. We trained two different HOG detectors using Daimler training samples and INRIA training samples. The steps to train a cascade of classifiers are as follow.

First, we need to create a vector file that consists of all positive training samples. For that, we need a text file containing information on the positive training samples. The text file will take the following form.

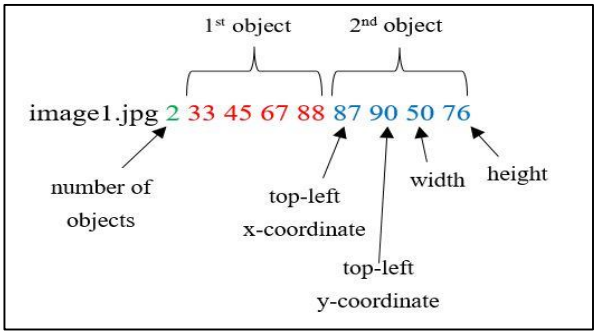

Figure 6.1: Text file format for list of positive samples

With the text file, use *opencv_createsamples.exe* to create a vector file containing the positive samples. Next, create another text file listing only the names of the negative samples. With the vector file and negative text file, use *opencv_traincascade.exe* to train a cascade classifier. When using *opencv_traincascade.exe*, specify the appropriate training parameters such as number of stages, feature type, number of positive samples, number of negative samples, etc.

Following the above steps, we trained two different HOG feature detectors using Daimler training samples and INRIA training samples. Daimler training samples consist of 15,660 positive samples and 6,744 negative samples. On the other hand, INRIA training samples consist of 2,416 positive samples and 1,218 negative samples. Both training samples cover pedestrians from all angles.

In the same manner, send the raw detections to the tracker. The tracker predicts the location of the pedestrians using Kalman filter and stores the output in a json format.

CHAPTER 7: RESULTS AND FINDINGS

7-1 Detection Results

Since we trained two different pedestrian detectors in the second experiment, we need to determine which detector has a better detection performance. Each detector is a cascade of classifiers trained to recognize the HOG features of a human shape. By testing the two detectors on the Town Centre video frames, we have the following ROC plot.

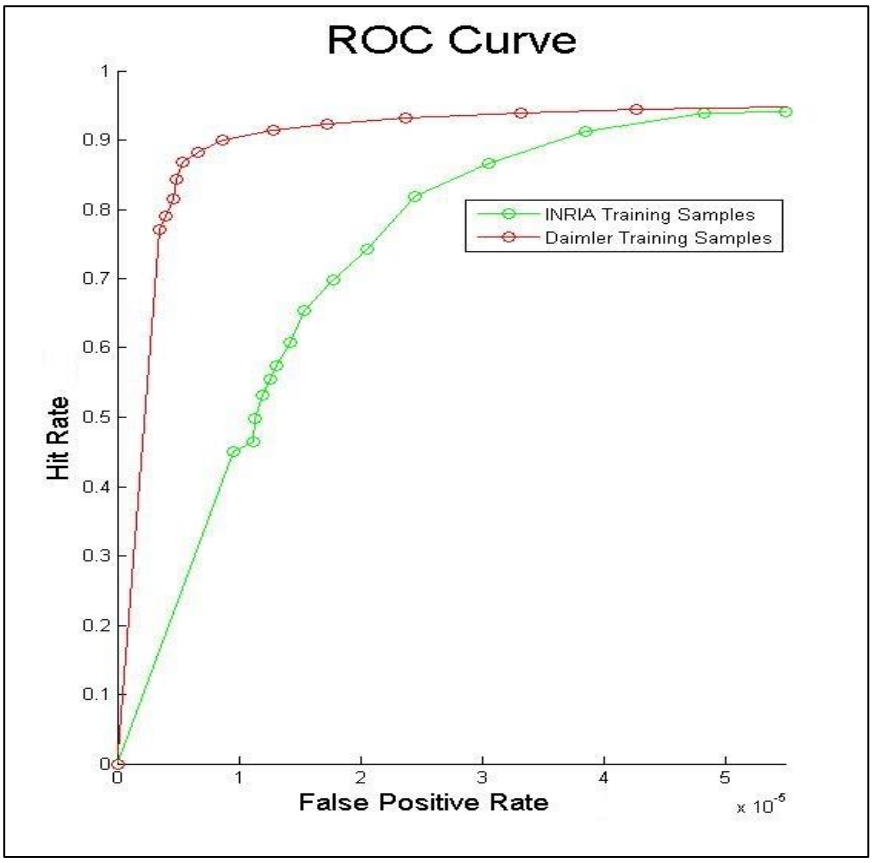

Figure 7.1: ROC curve for Daimler detector and INRIA detector

From Figure 7.1, both curves show an increase in the false positive rate when the hit rate increases. At a specific hit rate, the detector fed with Daimler training samples shows a lower false positive rate as compared to the detector fed with INRIA training samples. The Daimler detector reaches a hit rate of 90% much earlier and at a lower false positive rate as compared to the INRIA detector. As mentioned in the earlier section, the curve that is closer to the top left corner has a better discriminative ability. It means that, the detector trained with Daimler samples outperforms the detector trained with INRIA samples. Considering

that the Daimler detector was trained with more positive and negative images as compared to INRIA detector, we expect the Daimler detector to have a better detection performance. Only with large training samples, we can cover a variety of angles and possibilities, resulting in a more robust detector. Thus in the second experiment, we will use the raw detections from Daimler detector as an input to Kalman filter since it has a better performance as compared to the INRIA detector.

Figure 7.2: Raw detections

7-2 Tracking Results

For every raw detection, the centroid is computed and send to Kalman filter with the height and width. Using the information provided by the detector, Kalman filter estimates the location of the pedestrians *(blue bounding box)* as shown in Figure 7.3.

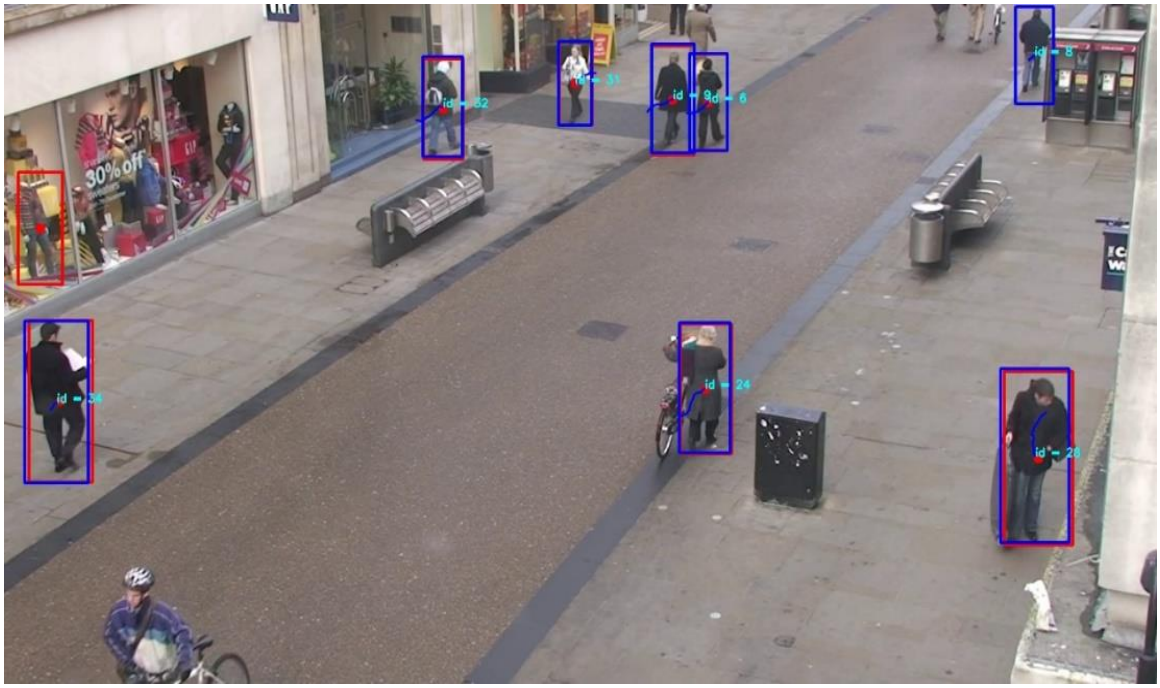

Figure 7.3: Tracking

Each pedestrian is assigned a unique identity for tracking purposes. Continuing in this manner, Kalman filter will continue to predict the location of the pedestrians in consecutive frames. For cases where the detector fails to provide information on the current location of the pedestrian, the filter will estimate the location of the pedestrian based on his/her previous location.

On the other hand if the detector is able to provide information on the location of the pedestrian, the filter will use this information to correct and refine its prediction. However this self-correcting mechanism has its limitation.

Since the filter has a degree of dependence on the information provided by the detector, it is important to ensure that the detector gives reliable detection results especially in the early stages of a Kalman cycle. Otherwise, the filter tracks the wrong object and this affects the overall tracking performance. Therefore, our algorithm only starts tracking if the detector is able to consistently detect the pedestrian for a few times in the early stage.

With this approach, our false positive rate is reduced by a significant amount as compared to our early results. In addition, it is assumed that false positives always appear at the same location throughout the video. The following is a comparison of our result with others using the CLEAR MOT metrics.

Table 7.1: Comparison of the tracking results using Town Centre data set

Algorithm	**MOTP**	**MOTA**	**Detection (sec/frame)**
Benfold & Reid	77.08 %	66.31 %	1.2
Dehghan, et al.	71.93 % (est.)	75.59 % (est.)	-
Izadinia, et al.	71.60 % (est.)	75.70 % (est.)	-
Dalal & Triggs SVM detector + proposed tracking algorithm (*Experiment 1*)	67.26 %	50.61%	4.5
Our trained boosted detector + proposed tracking algorithm *(Experiment 2)*	69.38 %	**67.07 %**	**0.76**

From Table 7.1, the algorithm proposed by Benfold and Reid (2011) achieved the highest MOTP between the four algorithms. With a Kanade-Lucas-Tomasi (KLT) approach and a Markov- Chain Monte-Carlo Data Association (MCMCDA) technique, they obtained a MOTP of 77.08%. On the contrary, Izadinia and his team achieved the highest MOTA which is 75.70% by constraining pedestrian tracking by parts tracking. Dehghan and his team (2014) obtained similar results to Izadinia's team (2012) by using a part-based human detector and a data association method called the Generalized Minimum Clique Graphs.

However the MOTP and MOTA values as reported in Deghan's paper and Izadinia's paper, are estimates only as we could not obtain their tracking output to run it on our evaluation algorithm. On the other hand, we can obtain the tracking output for Benfold and Reid's algorithm. Hence their MOTP and MOTA values in Table 7.1 are reported based on our evaluation algorithm. For a fair comparison, we will only compare our experiments with Benfold and Reid's results.

Comparing experiment 1 and 2, experiment 2 shows more promising results. By using a cascade of boosted classifiers for detection in experiment 2, our detection time was

shortened from approximately 4.5 to 0.76 seconds/frame as compared to using the SVM detector by Dalal and Triggs (2005) in experiment 1. The MOTP and MOTA values reported for experiment 2 are also better than experiment 1. Thus, we will only compare our experiment 2 results with Benfold and Reid's results.

In experiment 2, our algorithm had achieved a MOTA of 67.07% which is about 1% higher with better detection time as compared to Benfold and Reid. Our algorithm can also track pedestrians moving towards various directions in a semi-crowded environment as shown in the following figure.

Figure 7.4: Stable tracking in a semi-crowded environment

The measures used in our tracking evaluation are listed in the following table.

Table 7.2: Comparison of measures used in tracking evaluation

Measures	**Benfold & Reid**	**Proposed Algorithm** *(Experiment 2)*
Ground truths	71,460	71,460
False positives	10,515	**5,273**
Misses	13,188	17,915
Mismatches	370	**346**
Recoverable mismatches	331	316
Non-recoverable mismatches	157	111
Correspondences	58,272	53,545

From Table 7.2, our algorithm shows a lower number of false positives as compared to Benfold and Reid's algorithm. Our proposed algorithm has reduced the number of false positives by half. In conjunction, our algorithm also has a lower number of mismatches. Out of the 71,460 ground truths, our algorithm found only 53,545 correspondences between hypotheses and ground truths while Benfold and Reid's algorithm found a total of 58,272 correspondences.

In other words, our algorithm suffers from 25% misses while the researchers' algorithm only suffers from 18% misses which is the downside of our algorithm. The high number of misses affects our MOTP value, which is a measure of the precise location of the pedestrian. To achieve a significant improvement in our tracking results, we need to reduce the number of misses while keeping the number of false positives and mismatches constant. The following is an error analysis on the factors affecting the number of misses.

7-3 Error Analysis

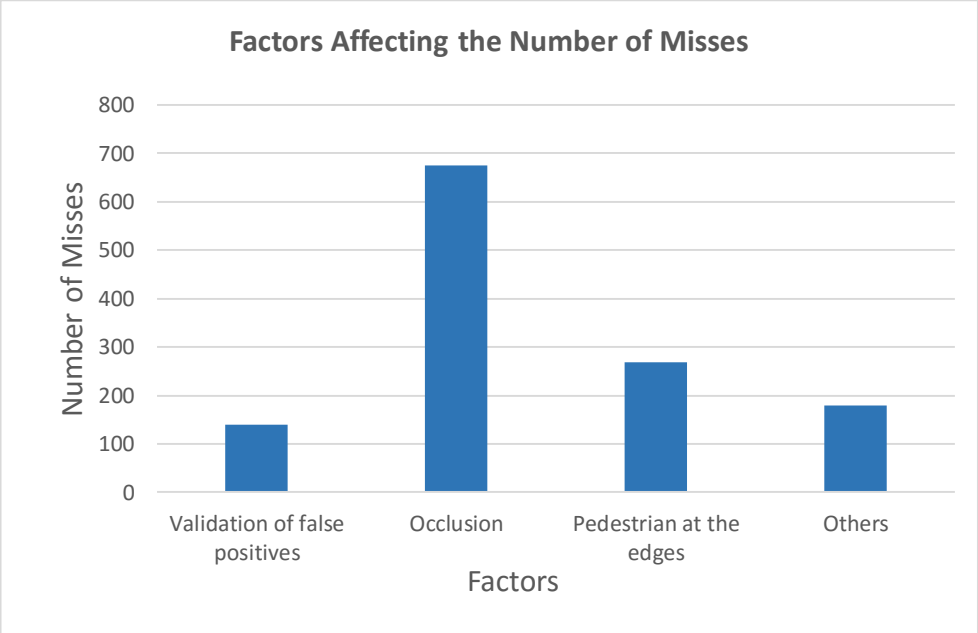

Figure 7.5: Error analysis on the factors affecting the number of misses

Figure 7.5 tells us that occlusion is one of the factors that affects the number of misses in our algorithm. It is because there is always a high degree of occlusion between pedestrians in a semi-crowded environment. Additionally the location of the pedestrian at the edges of the frame, either too near or too far away from the camera also affects the number of misses. Apart from these two factors, there are also other factors such as our validation check for false positives which also has an impact on the number of misses. Comparing all these factors, we observe that occlusion is the main factor that affects our tracking results. Thus to improve our results, our algorithm must be able to handle occlusion. This will be discussed further in the following section.

CHAPTER 8: CONCLUSION AND FUTURE WORK

8-1 Conclusion

At the end of this project, we have developed a pedestrian detection and tracking algorithm. We have shown that a cascade of boosted classifiers outperforms a SVM classifier. In terms of speed, the cascade has a shorter detection time per frame as compared to the SVM classifier in the OpenCV library. It is because the architecture of the cascade enables the negative sub-windows to be eliminated quickly for a fast detection. In addition, the tracking results using our trained cascade are better as compared to using the SVM pedestrian classifier in OpenCV library.

We have also learned that supplying more positive and negative training samples, we can have a detector with better detection performance. It is because a large number of training samples can account for more variation in the shapes and postures of the human body. Without doubt, a detector trained with more training samples will outperform a detector trained with less training samples. In our case, the Daimler detector with more training samples outperforms the INRIA detector.

Although our algorithm in experiment 2 has outperformed Benfold and Reid's algorithm in terms of speed and MOTA value, our MOTP value still falls behind by 7.7%. Our algorithm shows a better MOTA value because of the less number of false positives and mismatches as compared to the researchers'. However, it can be improved if we are able to reduce the number of misses. The large number of misses has affected our MOTP and MOTA values to some extent.

8-2 Future Work

As mentioned earlier, the high number of misses as reported in Table 7.2, is due to occlusion. Therefore to improve our results, one should try part-based detection to reduce the effect of occlusion on our tracking results. It is because when a part of the body is occluded, a part-based detector can still detect the partially occluded body but a full body detector will fail.

For instance Deghan, Idrees, Zamir and Shah (2014) had used a part-based detector that can handle occlusion in a crowded environment. Besides that, Benfold and Reid (2011) also used a head detector to estimate the full body region.

Hence, one should try to use a part-based detector instead of a full body detector for pedestrian detection and tracking especially in crowded environments.

REFERENCES

Athanesious, J. J. and Suresh, P., 2012. Systematic Survey on Object Tracking Methods in Video. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET).* [online] Available at: <http://ijarcet.org/wp-content/uploads/IJARCET-VOL-1-ISSUE-8-242-247.pdf> [Accessed 1 August 2015].

Benfold, B. and Reid, I., 2011. Stable Multi-Target Tracking in Real-Time Surveillance Video. *Computer Vision and Pattern Recognition (CVPR) 2011 IEEE Conference.* Providence, RI, 20-25 June, 2011. Colorado Springs: IEEE. Available at: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5995667&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5995667> [Accessed 12 July 2015].

Bernardin, K. and Stiefelhagen, R., 2008. Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics. *EURASIP Journal on Image and Video Processing.* [online] Available at: <http://jivp.eurasipjournals.com/content/pdf/1687-5281-2008-246309.pdf> [Accessed 10 July 2015].

Choi, B., n.d. Survival Manual for Statistical Analysis. [online]. Available at: <http://wwww.cbgstat.com/v2/method_ROC_curve_MedCalc/ROC_curve_MedCalc.php> [Accessed 23 October 2015].

Dalal, N. and Triggs, B., 2005, June. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* (Vol. 1, pp. 886-893). IEEE. Available at: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1467360&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1467360> [Accessed 10 August 2015].

Dehghan, A., Idrees, H., Zamir, A. R. and Shah, M., 2014. Automatic Detection and Tracking of Pedestrians in Videos with Various Crowd Densities. [online] Available at: <http://www.springer.com/cda/content/document/cda_downloaddocument/9783319024462-c1.pdf?SGWID=0-0-45-1445427-p175479038> [Accessed 12 July 2015].

Han, B. et al., 2009. Tracking of Multiple Objects under Partial Occlusion. *SPIE Defense, Security, and Sensing*. Baltimore, MD, United States, 23-27 April, 2012. Baltimore: SPIE. Available at: <http://www.wu.ece.ufl.edu/mypapers/trackingSPIE09.pdf > [Accessed 16 August 2015].

Izadinia, H., Saleemi, I., Li, W. and Shah, M., 2012. Multiple People Multiple Parts Tracker. *European Conference on Computer Vision 2012*. Florence, Italy, 7-13 October, 2012. Orlando, Florida: University of Central Florida. Available at: < http://homes.cs.washington.edu/~izadinia/files/MPMPT-ECCV12.pdf> [Accessed 3 August 2015].

Jiang, Z. et al., 2010. Multiple Pedestrian Tracking using Colour and Motion Models. [online] Available at: <http://www.csse.uwa.edu.au/~du/ps/Jiang-et-al-DICTA10.pdf> [Accessed 12 July 2015].

Jin, Y., Soh, W.S., Motani, M. and Wong, W.C., 2013. A robust indoor pedestrian tracking system with sparse infrastructure support. *Mobile Computing, IEEE Transactions on*, *12*(7), pp.1392-1403. Available at: <https://www.ece.nus.edu.sg/stfpage/elesohws/TMC_SparseTrack.pdf> [Accessed 6 July 2015].

Munkres, J., 1957. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics,* [online] 5(1), pp. 32-38. Available through: UC Davis Mathematics website < https://www.math.ucdavis.edu/~saito/data/emd/munkres.pdf> [Accessed 5 July 2015].

Nagendran, A., Dheivasenathipathy, N., Nair, R. V. and Sharma, V., 2014. Recognition and Tracking Moving Objects Using Moving Camera in Complex Scenes. *International Journal of Computer Science, Engineering and Applications (IJCSEA)*. [online] Available at: <http://airccse.org/journal/ijcsea/papers/4214ijcsea03.pdf> [Accessed 8 July 2015].

Osuna, E., Freund, R. and Girosi, F., 1997, June. Training support vector machines: an application to face detection. In *Computer vision and pattern recognition, 1997. Proceedings., 1997 IEEE computer society conference on*(pp. 130-136). IEEE. Available at: <http://web.mit.edu/rfreund/www/10.1.1.9.6021.pdf> [Accessed 12 March 2016].

Rakibe, R.S. and Patil, B.D., 2013. Background subtraction algorithm based human motion detection. *International Journal of scientific and research publications*, [online] *3*(5). Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.414.5782&rep=rep1&type=pdf> [Accessed 16 February 2016].

Shantaiya, S., Verma, K. and Mehta, K., 2013. A Survey on Approaches of Object Detection. *International Journal of Computer Applications*, [online] *65*(18). Available at: <http://search.proquest.com/openview/afaa3ddb217f4e5e343a55837681c062/1?pq-origsite=gscholar> [Accessed 10 January 2016].

Skoglar, P., Orguner, U., Törnqvist, D. and Gustafsson, F., 2012. Pedestrian tracking with an infrared sensor using road network information. *Journal on Advances in Signal Processing 2012*. [online] Available at: <http://asp.eurasipjournals.com/content/2012/1/26#refs> [Accessed 14 July 2015].

Tian, X., Bao, H., Xu, C. and Wang, B., 2013. Pedestrian Detection Algorithm based on Local Color Parallel Similarity Features. *International Journal on Smart Sensing and Intelligent Systems*, [online] *6*(5), pp.1869-1890. Available at: <http://s2is.org/Issues/v6/n5/papers/paper3.pdf> [Accessed 8 January 2016].

Vijayakumar, M., 2014. *Real time robust human detection and tracking.* [pdf] Chennai: Madras Institute of Technology, Anna University. Available at: <https://www.academia.edu/10084929/Real_Time_Human_Detection_And_Tracking_System> [Accessed 28 July 2015].

Viola, P. and Jones, M., 2001. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* (Vol. 1, pp. I-511). IEEE. Available at: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=990517&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D990517> [Accessed 25 November 2015].

Welch, G. and Bishop, G., 2006. An Introduction to the Kalman Filter. [online] Available at: < https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf> [Accessed 10 July 2015].

Yilmaz, A., Javed, O. and Shah, M., 2006. Object Tracking: A Survey. *Journal ACM Computing Surveys*, [e-journal] 38(4). Available through: ACM Digital Library website < http://dl.acm.org/citation.cfm?id=1177355> [Accessed 8 August 2015].

APPENDICES

Appendix A: Training Parameters

Table A: Training parameters in experiment 2

Parameters	Daimler Training Samples	INRIA Training Samples
Number of Positive	14,000	2,170
Number of Negative	14,300	2,210
Number of Stages	17	20
Feature Type	HOG	HOG
Width	48	64
Height	96	128

Appendix B: Source Code

```
//in main.cpp
#include "stdafx.h"
#include <stdlib.h>
#include "opencv2/opencv.hpp"
#include "opencv/cv.h"
#include <opencv2/highgui/highgui.hpp>
#include "Ctracker.h"
#include <iostream>
#include <vector>
#include "json.h"
#include "json.cpp"
#include <fstream>
#include <direct.h>
using std::string;
using std::cout;
using std::cerr;
using std::endl;

using namespace cv;
using namespace std;



int main(int ac, char** av)
{
	const float eachframeTime = 0.04f;  // the town centre video has 25 frames per
second, 1/25=0.04s (time interval between each frame)

	vector<Rect> centers1;

	Mat img, frame;
	FILE* f = 0;
	double topLeft, bottomRight;
	char _filename[1024];
	char directory[128];
	long frame_count = 0;
	char directoryNew[128];
	char trackId[128];
	bool should_stop = false;

	string videoName = "TownCentreXVID.avi";
	string cascadeName = "cascade_14000_16.xml";
	CascadeClassifier hog_cascade;
	VideoCapture cap(videoName);

	if (!cap.isOpened())
	{
		std::cerr << "ERROR: Could not open video " << videoName << std::endl;
		return 1;
	}

	CTracker tracker1(0.2, 0.5, 70.0, 15, 20);

	json::Array arr;
	json::Object obj;
	json::Array subframe_arr;

	_mkdir("processed"); //store the output frames in this folder

	while (!should_stop) // start retrieving video
	{
```

```
		cap >> frame; //get a new frame from the video
		centers1.clear();

		if (frame.empty() || (frame_count == 4501)) //arrived to the end of a
short video or to the end of a 3minute video
		{
			should_stop = true;
			break;
		}


		sprintf_s(directory, "frame_%06ld.jpg", frame_count);
		strcpy_s(_filename, directory);
		img = frame;

		if (!hog_cascade.load(cascadeName)) //load cascade
		{
			printf("--(!)Error loading\n"); return -1;
		}

		//HOGDescriptor hog;
		//hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());

		double t;
		for (;;)
		{
			char* filename = _filename;
			if (f)
			{
				if (!fgets(filename, (int)sizeof(_filename) - 2, f))
					break;

				if (filename[0] == '#')
					continue;
				int l = (int)strlen(filename);
				while (l > 0 && isspace(filename[l - 1]))
					--l;
				filename[l] = '\0';
				img = imread(filename);
			}

			if (!img.data)
				continue;

			fflush(stdout);
			vector<Rect> found, found_filtered;

			t = (double)getTickCount();

			//hog.detectMultiScale(img, found, 0.15, Size(8, 8), Size(32, 32),
1.05, 2);

			hog_cascade.detectMultiScale(img, found, 1.1,3, 0, Size(50, 120),
Size(170, 350));

			t = (double)getTickCount() - t;
			printf("detection time = %gms\n", t*1000. /
cv::getTickFrequency());
			size_t i, j;

			for (i = 0; i < found.size(); i++)
			{
				Rect r = found[i];
				for (j = 0; j < found.size(); j++)
					if (j != i && (r & found[j]) == r)
						break;
```

```
                        if (j == found.size())
                                found_filtered.push_back(r);
                }

                for (i = 0; i < found_filtered.size(); i++)
                {
                        Rect r = found_filtered[i];
                        r.x += cvRound(r.width*0.15);
                        r.width = cvRound(r.width*0.68);
                        r.y += cvRound(r.height*0.07);
                        r.height = cvRound(r.height*0.85);
                        //rectangle(img, r.tl(), r.br(), cv::Scalar(0, 0, 255), 3);
//show detection
                        Point center;
                        Rect rtemp;
                        rtemp.x = r.x;
                        rtemp.y = r.y;
                        rtemp.width = r.width;
                        rtemp.height = r.height;
                        center = Point(r.x + (r.width / 2), r.y + (r.height / 2));
                        //circle(img, center, 8, Scalar(0, 0, 255), -1, 1, 0);
//show detection
                        centers1.push_back(rtemp);

                }

                int c = waitKey(0) & 255;
                if (c == 'q' || c == 'Q' || !f)
                        break;
        }

        json::Array hypotheses_arr;
        tracker1.Update(centers1);
        if (centers1.size()>0)
                for (int i = 0; i<tracker1.tracks.size(); i++)
                {
                        if (tracker1.tracks[i]->trace.size()>3 &&
(!tracker1.tracks[i]->false_pos))  //  false positive check
                        {
                                for (int j = 0; j<tracker1.tracks[i]->trace.size()
- 1; j++)
                                {
                                        line(img, tracker1.tracks[i]->trace[j],
tracker1.tracks[i]->trace[j + 1], Scalar(255, 0, 0), 2, CV_AA);
                                }
                                sprintf_s(trackId, "id = %d",
(int)tracker1.tracks[i]->track_id);
                                // bounding rec for tracking

                                Rect trackRec;
                                trackRec.x = tracker1.tracks[i]-
>trace[tracker1.tracks[i]->trace.size() - 1].x - (tracker1.tracks[i]->rec.width / 2.0);
//topLeft point
                                trackRec.y = tracker1.tracks[i]-
>trace[tracker1.tracks[i]->trace.size() - 1].y - (tracker1.tracks[i]->rec.height / 2.0);
                                trackRec.width = tracker1.tracks[i]->rec.width;
                                trackRec.height = tracker1.tracks[i]->rec.height;
                                printf("Centroid (%f, %f)",
(float)tracker1.tracks[i]->trace[tracker1.tracks[i]->trace.size() - 1].x,
(float)tracker1.tracks[i]->trace[tracker1.tracks[i]->trace.size() - 1].y);
                                printf("  \nWidth: %d Height: %d upLeft:
(%d,%d)\n\n", trackRec.width, trackRec.height, trackRec.x, trackRec.y);
                                rectangle(img, trackRec.tl(), trackRec.br(),
cv::Scalar(255, 0, 0), 3);
```

```
                                    putText(img, trackId, tracker1.tracks[i]-
>trace[tracker1.tracks[i]->trace.size() - 1], FONT_HERSHEY_SIMPLEX, 0.6, cv::Scalar(255,
255, 0), 2);

                                    json::Object hypotheses_obj;

                                    hypotheses_obj["height"] = trackRec.height;
                                    hypotheses_obj["width"] = trackRec.width;

                                    sprintf_s(trackId, "%d", (int)tracker1.tracks[i]-
>track_id);

                                    hypotheses_obj["id"] = trackId;
                                    hypotheses_obj["y"] = trackRec.y;
                                    hypotheses_obj["x"] = trackRec.x;

                                    hypotheses_arr.push_back(hypotheses_obj);
                            }
                        }

                json::Object frame_object;
                frame_object["timestamp"] = ((float)frame_count)*eachframeTime;
                frame_object["num"] = (int)frame_count;
                frame_object["class"] = "frame";
                frame_object["hypotheses"] = hypotheses_arr;
                subframe_arr.push_back(frame_object);
                sprintf_s(directoryNew, "processed/frame_%06ld.jpg", frame_count);
                imwrite(directoryNew, img);
                printf("\nProcessed frame_%06ld.jpg\n", frame_count);
                frame_count++;
        }

        obj["frames"] = subframe_arr;
        obj["class"] = "video";
        obj["filename"] = "D:/TownCentreXVID.avi";

        arr.push_back(obj);
        std::string serialized_string = json::Serialize(arr);
        cout << serialized_string << endl;

        fstream file;
        file.open("hypotheses.json", ios::out);
        file << serialized_string;
        file.close();
        waitKey(30);
        return 0;

}
```