Distributed Camera Tracking Network

ECSE 6500: Distributed Systems and Sensor Networks

Usama Munir (661541154), Yogish Didgi (661538970), Zaid Bin Tariq (661677311)

Abstract

In this course project, we have implemented a distributed camera tracking for multiple agents. We have used two cameras for the implementation of the system but the system can easily be scaled to multiple cameras. Two cameras with some overlapping field of view are used to test our implementation. The user in field of view of camera 1 is assigned an identifier which is carried through when the same person enters camera 2 field of view. For this purpose, we have distributed and centralized processing of the incoming video feed. The centralized system is responsible for doing handover of the tagged person from one camera to another. We detect the agents using Histogram of oriented gradients algorithm in each frame with tracking of the each agent being done using MIL algorithm. For the consistency of the ID of the agent in both camera one and two, we get features of the tracked agents using SIFT and Histogram feature vector (HFV) algorithm to make a decision about IDs of the overlapping agent in the camera field of view of two cameras.

1 Introduction

The use of camera network for the purpose of tracking agents is an active research problem. The main use-case of this system can be in intelligent surveillance with little human intervention. Crowd Monitoring or tracking individuals in a crowded setting is another application. The commercial use of this system is possible in sports, for example, automatically tracking players on the field/court to display stats, IDs and many more.

The objective of this project is to track multiple agents across multiple cameras. We have used distributed processing for tracking agents for each camera in the camera network and centralized system for handover of an identifier between the central and local network. Figure 1 shows the general working of the system. We have camera video coming in as an input at each local camera node which will be processed to track the agents within the network with each agent having a unique global ID.

Multi-camera tracking of people is a hard problem to solve. In engineering literature it is also referred to as multi-camera pedestrian tracking. Researchers have proposed many solutions over the years to solve this problem, most of which are tailored towards specific scenarios. A popular method to track people in a multi-camera environment comes from [FBLF08]. In this work, researchers have been successful in using cameras distributed around a confined physical location to create a probability occupancy map of the people moving around within that area. Furthermore, they provide video datasets for a standardized evaluation of multi-camera tracking algorithms. One of these data sets has been used to test our method as well. The tracking results for this work are very accurate. However, it requires that the calibration information of the cameras be known before-hand.

Another work that implements distributed multi-target camera tracking uses a Consensus Kalman Filter to track the 3D location of a person in a network of distributed cameras with overlapping fields of view. Again, 3D state information of a person's location and speed are employed in this method that require careful calibration of all the cameras. Our work, in comparison, relies only on the 2D information present in the image frames of a distributed network of cameras. Therefore, it has the advantage of deployment to any arbitrary network of cameras without the need for calibration. To achieve this, we have relied on computer vision tracking algorithms, specifically the Multiple Instance Learning algorithm for tracking.

There are three main tracking algorithms used in video tracking of objects: 1) Tracking Learning Detection (TLD), 2) Kernalized Correlation filters (KCF) and 3) Multiple Instance Learning (MIL). There are advantages and disadvantages of using any of these tracking algorithms. While TLD [KMM12] is great at handing occlusions for tracking, it loses track of its target if it makes sudden

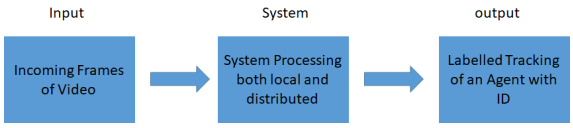

Figure 1: The general problem statement with the input as a camera feed in to the local camera network and output as the globally ID'ed agents with tracking

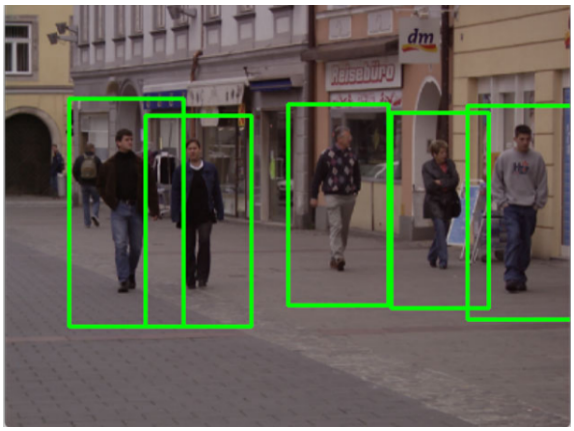

Figure 2: A sample output of the HOG pedestrian detection algorithm. It is used as the measurement tool in our method. Image from [Ros15]

abrupt movements, especially turns. Since, in a multi-camera environment, people can be expected to make turns or rotate about a position, we have cautioned against the use of TLD as a tracking algorithm. Compared to TLD, KCF [HCMB15] is much better at handling abrupt movements. It is also much faster than the MIL algorithm [BYB09]. This is because the KCF tracker combines the best of MIL and boost tracking to track an object. It is also the default tracker recommended among OpenCV users. However, it may lose track of objects when it trades off accuracy for speed. Considering, this we have decided to use the MIL tracking algorithm. We have explained the principles behind the working of this algorithm in section 2.3.

2 Methodology

Our tracking network has distributed processing both at the local camera level and at the central server. The aim of the local processing at each camera is to sense the environment (in this case making video) and then process that input video to identify agents in a frame (we detect persons every 10th frame). The identification of each agent is done using histogram of oriented gradients algorithm which is discussed in section 2.1. The identification of the pedestrian is necessary so that the identified part can be used as an input to the MIL tacking algorithm. Figure 2 shows the identified agents by the HOG algorithm. The rectangle bounding boxes are then used as an input to the MIL tracker.

The local cameras also assign a local ID to each agent in their own respective field of vision. This ID is used for mapping same agents in different camera field of vision which helps the central server in assigning a global ID to each agent in the network. The central server after receiving the information from the local cameras, applies a feature detection algorithm to map each agent and check for similar agent in other cameras' field of view. In the proceeding sections, we are going to provide the details for detection and tracking of agents.

2.1 Pedestrian Detection

The detection of a pedestrian is the most important and the very first step for the overall working of the tracking network. The MIL tracking algorithm requires a initial section of image, which it can use as a ground truth for tracking. We have used Histogram of oriented gradients (HOG) in order to detect a pedestrian. The output of the HOG is a set of bounding boxes encompassing the

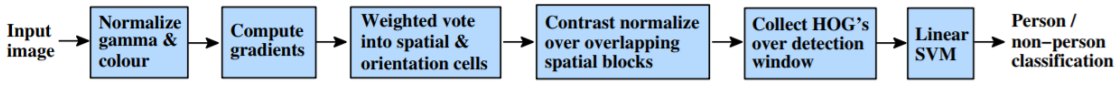

Figure 3: HOG block diagram [DT05].

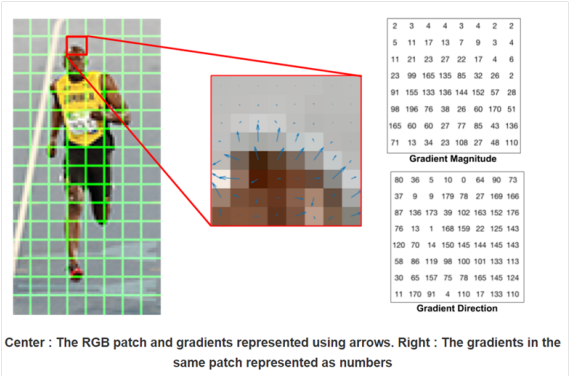

Figure 4: Dividing an image into cells [Mal16].

pedestrians found in that image. These bounding boxes are characterized by the pixel coordinates of the top-left corner and its width and height. The details of the HOG algorithm is given in the following section.

2.2 Histogram of Oriented Gradients (HOG)

To track any object in an image, in this case a person, we must be able to detect it first. The histogram of oriented gradients (HOG for short), by Dalal et. al, is a reliable method to train an object classifier [DT05]. A pedestrian can be characterized by the silhouette. HOG descriptor is based on this idea. The pedestrian is represented by a distribution of local gradients and these gradients represent the shape of the person. There are two important parts to detecting the pedestrian; HOG feature descriptor and support vector machine (SVM) classifier.

2.2.1 HOG Feature Descriptor

The image in which we want to detect the presence of a pedestrian is passed through a gamma correction module for contrast enhancement. It is then scanned using a sliding window of size 64x128 pixels. The window size chosen depends on the application and the environment. Gradients are computed for each window. The window is then divided into several cells of 8x8 pixels (refer 4). For each cell, the gradient magnitude and direction is represented by a 8x8x2 = 128 length vector. The gradient direction is restricted to lie in the range of 0 to 180 degrees. The gradients are collected into a histogram of 9 bins, with each bin catering to 20 degrees of the gradient orientation. The histogram bin is selected based on the gradient direction and the value added to the bin depends on the gradient magnitude. The resulting 9x1 histogram looks like figure 5. Four such neighboring cells are clubbed together into a 36x1 vector and normalized by its magnitude to account for intensity variations. After this processing, we concatenate all such histogram vectors into a single 3780 length feature vector. Note that the above process can be carried out at multiple scales of the image in order to handle scale variations in the image.

2.2.2 SVM Classifier

Now that the image patch is represented by a feature vector, it is necessary to decide whether it represents a pedestrian or not. This is done by classifying this feature vector using a SVM. The OpenCV implementation of pedestrian detection was trained with the INRIA dataset. The result of the classification is a boolean decision on whether or not the current image patch represents a human (refer figure 6). If a human is detected, the location of the image patch and its width and height is saved.

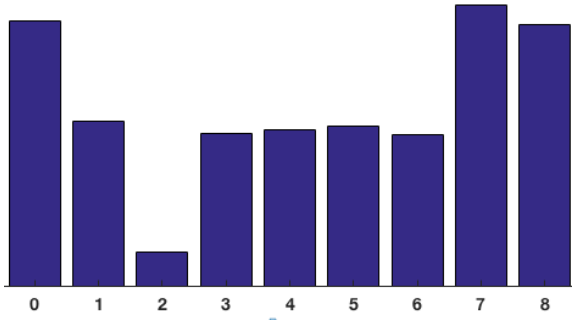

Figure 5: Typical histogram feature vector [Mal16].

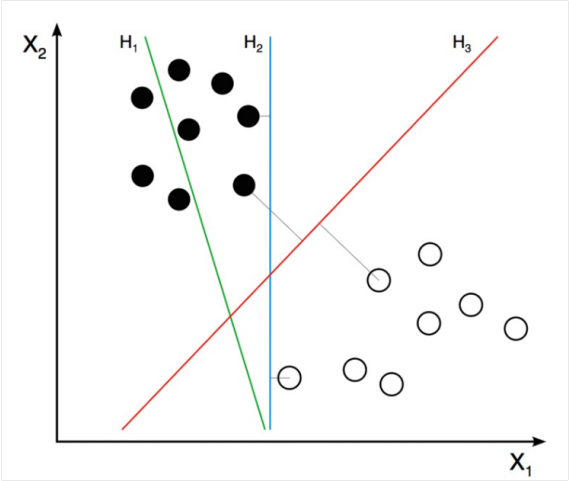

Figure 6: Illustration of linear SVM classification. Figure from [Mal16]

2.3 Tracking: Multiple instance learning (MIL)

Our tracking algorithm of choice for this project is the MIL 'Online' tracking algorithm from [BYB09]. The premise of the algorithm is to track any object in a video, given its location in the first frame. The location, for instance, can be provided in the shape of an image patch bordered by a rectangular bounding box. This initial input is labeled as a positive example. A positive example represents a training data point corresponding to the object we want to track in the image. We also crop out some negative examples around the positive example that correspond to the background. These negative patches are taken near the object to be tracked in a given radius. Now, we can train a discriminative classifier on the training data of positive and negative patches that we have collected in the active frame. This is illustrated in figure 7(A).

When the next frame comes, we can use the old tracker location to define a region around it for us to apply the classifier on. This greedy approach is dependent on the frame rate of the video and somewhat on the speed of the object being tracked. The classifier would then provide us new estimates of the location of the object within our defined region with some probability. We can then take the maximum likelihood of the results to find our new tracker location. If we denote the tracker location at time step t, as $l_t{}^*$ and let $l(x)$ be the location of image patch x, we can mathematically write this greedy approach as:

$$l_t{}^* = l(\underset{x \in \mathcal{X}^s}{\operatorname{argmax}}\, p(y|x)) \tag{1}$$

where $\mathcal{X}^s$ is the set of all cropped out image patches. While this works for the first few frames, sudden movements from the object, like turning of the head, may throw off the classifier. Therefore, our tracker would lose track of the object. Now, instead of using a few training examples, we can take multiple positive and negative examples in each frame. This is illustrated well in figure 7(B).

Adding more training examples definitely helps the tracking but it might confuse the classifier during online-learning where our labels for the training image patches may not be accurate.

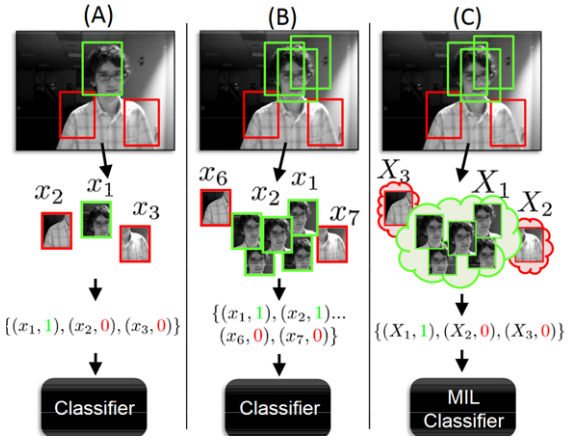

Figure 7: **Updating a discriminative appearance model:** .(A) Using a single positive image patch to update a traditional discriminative classifier. The positive image patch chosen does not capture the object perfectly. (B) Using several positive image patches to update a traditional discriminative classifier. This can confuse the classifier causing poor performance. (C) Using one positive bag consisting of several image patches to update a MIL classifier Figure and caption from [BYB09]

Therefore, the final step in this algorithm is to divide examples into bags where the whole bag is labeled as positive if there is at least one positive example (See figure 7(C)). This solves the problem of confusing the tracker and we can continue onwards by repeating our three step process of (1) generating positive and negative examples in bags, (2) training a classifier and (3) use the maximum likelihood estimate to find the new tracker location in the active frame.

2.4 Central Server

The central server needs to assign a unique global ID to every agent within the network. Figure 8 and 9 shows one of the frames received by the central server (The central server in fact gets the rectangular bounded objects identified by each local camera node). In order to assign a unique ID, the server needs to distinguish between the multiple agents and also match the same agents between the two cameras. This is problematic because as seen in the figure 8 and 9, the perspective of the camera 2 in figure 9 is different from that of camera 1 in figure 8. Hence the person in black shirt will be difficult to map to the same person in camera 1. Furthermore, because of multiple agents, we also need to distinguish between person in pink and black shirt. In our project, we have used both SIFT [Low04] and color histogram for generating feature vector descriptor and we have found out that for our purposes, the use of histogram feature vector works better. The following subsection will summarize the techniques of SIFT and Histogram feature vector.

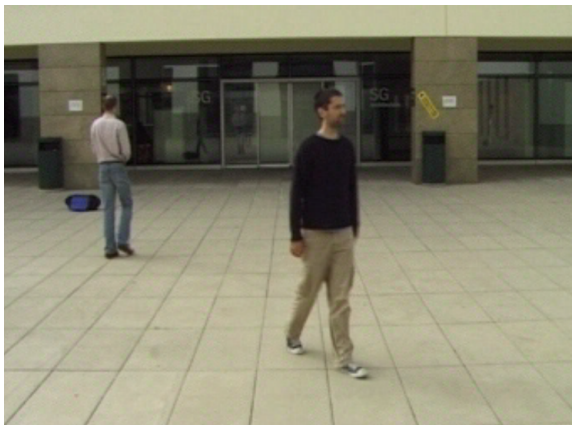

Figure 8: A frame showing the perspective of one of the cameras (camera 1). The output of the HOG is used as query image for feature detection for mapping at central server. [BFTF11]

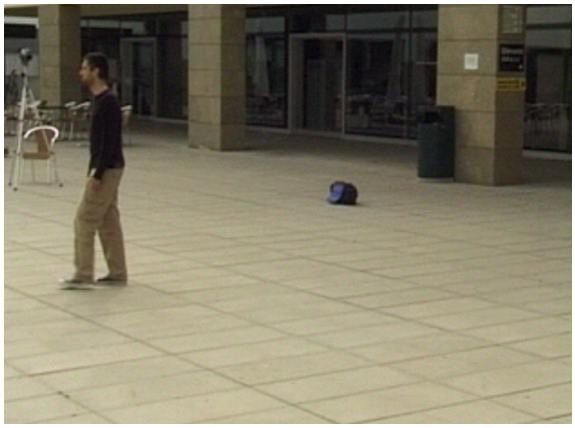

Figure 9: A frame showing the perspective of other camera (camera 2). The output of the HOG's i^{th} object is used as a test image for query to check whether similar agents are present in the other camera field of vision [BFTF11]

2.4.1 Scale-invariant feature transform (SIFT)

One of the aims of the project was to track an agent and assign them an ID for tracking. Note that this ID should be consistent for both camera 1 and camera 2. But each camera in the network has its own way of assigning the IDs. This is because a particular agent can enter the field of vision of another camera at a random time which means the dictionary at any n_{th} camera is going to be populated as and when an agent enters the field of view, which means that the local IDs of every agent will not be consistent across all cameras in the network. We have used SIFT algorithm to assign consistent ID at the central server. SIFT is an algorithm that helps to generate unique features of an object which can be used to detect an object. For solving the problem of consistency of the IDs, we use SIFT to get the features of the agent in a camera and try to match it with the agents in the other camera so that we can differentiate between the overlapping agent for both camera one and two. In this way the central server is able to decide on the agent appearing in both cameras' field of view, thus making the decision of assigning same ID to a particular agent. SIFT algorithm constructs a scale space which is for the internal representation of the image to ensure scale invariance. It then uses the difference of maxima and minima in the difference of Gaussian image to get the key points. Some of the bad key points like edges are removed using Harris Corner Detector [Hs88]. The orientation of the remaining features is then changed and then one more representation of the features is generated for better feature. More on SIFT can be found here [Low04].

2.5 Histogram Feature Vector for feature detection(HFV)

In the central server, we need to distinguish one person from another. In order to do this, we need to rely on the appearance of the person. The appearance is well characterized by the distribution of the image intensity, in other words, a histogram. The color image consists of three channels; red, green and blue. And we are using 8-bit quantization levels to represent the intensity at a pixel for each channel. So, there are 256 possible intensity levels. A histogram is constructed for each channel. For each pixel, we identify the bin it belongs to and add one to that bin. There are generally 256 bins. So, in a typical histogram plot, x-axis represents the possible pixel intensity and y-axis represents the number of pixels in the image with that intensity. The number of bins is quantized to 64 in order to reduce the noise. The idea is we want to capture the trend of the histogram and aren't really interested in the value for each bin. This results in a 64x1 vector for each channel. The histogram vectors from all the three channels are concatenated to form a 192x1 length feature vector.

Now, that we have the feature vector representing the person in the camera, we can compare it to the feature vector representing a person in another camera. The distance measure used is the Euclidean distance. If a person appears in the field of view of both the cameras, then in the central server, the euclidean distance between the agents representing the same person must be the the least amongst all and fall below a certain threshold, and must be higher when the agents

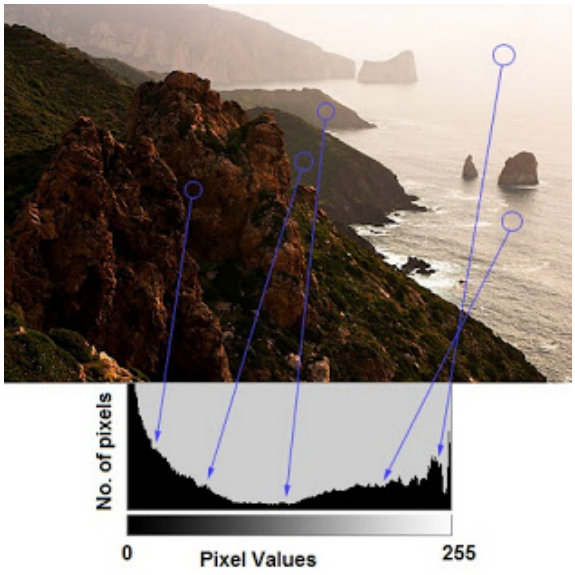

Figure 10: Sample histogram feature vector for an image [His15].

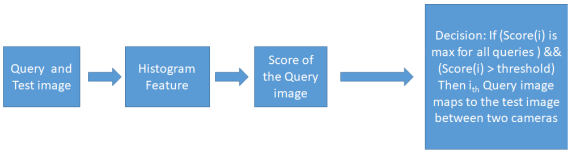

Figure 11: Central server decision making.

belong to different persons.

2.5.1 Central Server Decision Making

The algorithms described in section 2.5 and 2.4.1 are able to produce the feature vector which is then used to determine the similarity between different query images of agents with respect to the test image. Figure 11 shows the process of making the decision for matching agents between different cameras. The query image and the test image are fed into the algorithm in the previous subsection (SIFT or HFV). The algorithm outputs a score for each image. The central server will match an agent (lets say query image from one of the camera) with the test image (test agent from other camera), if its score is maximum compared to all other query while at the same time exceeding a predefined threshold. In this way are able to identify same agent in two different cameras' field of view.

3 Implementation and Evaluation

We have used openCV in Python to implement the system. The local camera network was simulated using a laptop running a Linux environment. We have used two datasets. One of them is available here [BFTF11]. We also collected the data set as presented later in this section.

The output of people detection code can be seen in figure 12. As can be seen in the video, multiple people in an image are easily detected and we draw a bounding box around each of the persons detected and assign a unique ID to each person. The ID assigned is local to that camera and doesn't relate to the global ID.

The tracking result for each of the cameras is shown in figures 13 and 14. Note that only certain key frames are shown here. As can be seen from the figure, the ID's assigned to agents doesn't change across the image frames, which implies successful tracking of those agents.

The implementation also handles tracking multiple agents in a frame. The result for such a scenario is shown in figure 15. As can be seen, the ID's assigned to the agents remain same across frames and doesn't get exchanged with other agents in the image.

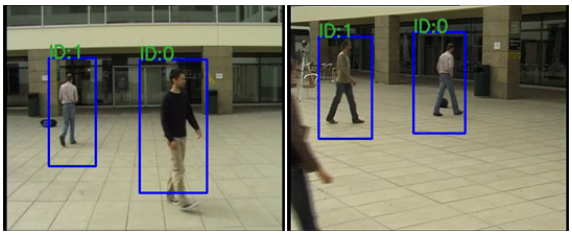

Figure 12: People detection output

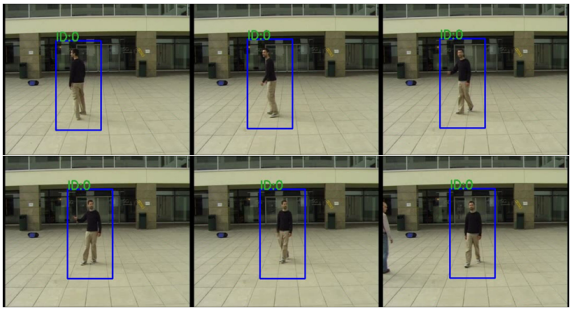

Figure 13: People tracking output for camera 1

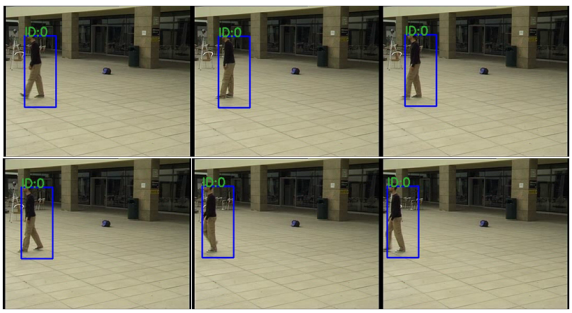

Figure 14: People tracking output for camera 2

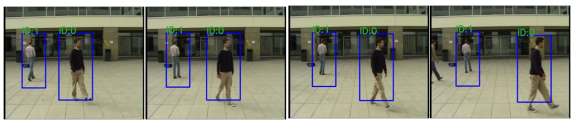

Figure 15: Tracking multiple people in camera 1

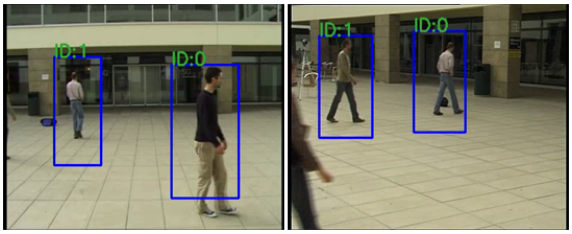

Figure 16: Before central server processing

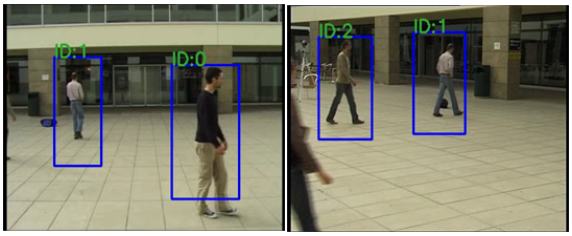

Figure 17: After central server processing

Figure 18: People tracking output for camera 1 for dataset 2

The result of tracking across multiple cameras before and after central server processing is shown in figures 16 and 17. In figure 16, we can see camera 1 output at a particular frame. The camera 2 output at that instant can be seen on the right side. Note here that the ID's assigned are local to each of the cameras. The result after central server processing is shown in figure 17. From the figure, we can see that, local ID-1 in camera-1 and local ID-0 in camera-2 represent the same person. Hence, the global ID assigned to that person is the same (ID-1) after processing.

The results for the video we shot ourselves is shown in figures 18 and 19. As can be seen from the figures, multiple people are detected without fail and are tracked properly across frames. The ID's assigned are same if the person appearing in the two cameras is the same. The person missed in figure 19, frame 4, is because we are not detecting persons near the image boundary. This was designed in this manner to handle shortcomings of the tracker.

4 Problems encountered and future work

The people detection code sometimes fails to detect a person in the image. The scenario is shown in figure 20. This is because the SVM we use for classification is trained on a different environment than the one we are using it in. We found that using gamma correction beforehand helped to some extent but didn't completely eliminate the problem.

The tracker we used sometimes drifted leading to the person not being tracked properly. The tracker would learn the features in the background and treat them as features to be tracked. This can be seen in figure 21. The bounding box did not resize properly to only cover the person. But rather it is tracking some features along the ground. Over several frames, even when the person is moving, the tracker drifts by continuing to track features in the background. Eventually, the person detection code detects the same person as a new agent and assigns a new ID. The tracker continues to track the old agent, even though there is no person present inside it.

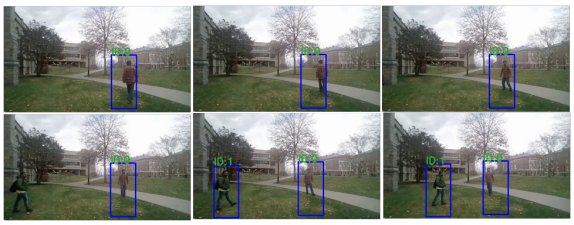

Figure 19: People tracking output for camera 2 for dataset 2

Figure 20: People detection failure case

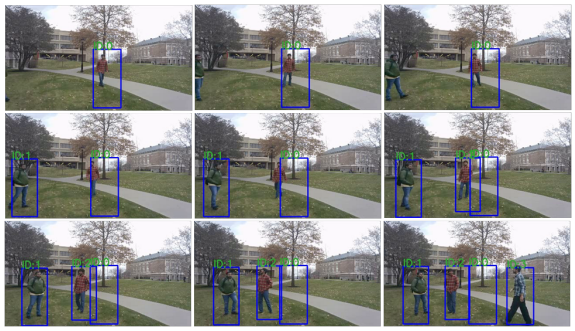

Figure 21: People tracking failure case

5 Conclusion

In this project, we have implemented a distributed camera tracking network using openCV in Python. The cameras at the local nodes process the incoming video in order to identify agents which is then used by the MIL tracking algorithm to track the multiple agents. For unique ID assignment for each agent within the camera tracking network, we have used SIFT and HFV for feature detection in the central server. Our final system design uses HFV because of relatively better results. In future work, we aim to make the feature detection of the system more robust while at the same time maintaining the fast working of the overall system.

References

[BFTF11] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua. Multiple Object Tracking using K-Shortest Paths Optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[BYB09] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 983–990. IEEE, 2009.

[DT05] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[FBLF08] Francois Fleuret, Jerome Berclaz, Richard Lengagne, and Pascal Fua. Multicamera people tracking with a probabilistic occupancy map. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):267–282, 2008.

[HCMB15] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015.

[His15] Opencv python histogram tutorial, 2015. Accessed: 2017-12-02.

[Hs88] C Harries and Mike steven. A combined corner and edge detection. *Alvey Vision Conference*, 1988.

[KMM12] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409–1422, 2012.

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *Internationa Journal of Computer Vision*, 2004.

[Mal16] Satya Mallick. Histogram of oriented gradients, 2016. Accessed: 2017-12-02.

[Ros15] Adrian Rosebrock. Pedestrian detection opencv, 2015. Accessed: 2017-12-02.