Automating Visual Sensor Networks

Mayank Kumar
Computer Science Department
Brown University
mayank@cs.brown.edu

Abstract

Automating surveillance system is an interesting research area with real world application. In our work we are building an automated surveillance system with no single point of failure, and one which provides the user with an abstract interface to make queries without the knowledge of the underlying sensors and their configuration. We have been working on the infrastructure needed to support this system and the protocols that will make the system run. We also researched different ways to describe and track objects seen by the sensors and came up with models that reinforce one descriptor with another as they become available.

Introduction

To build an automated reliable surveillance system is a very challenging issue. Ideally, we want the surveillance system to find and track "interesting" objects automatically without having any human intervention but allowing a certain degree of configurability. With the cost of surveillance cameras going down and their numbers constantly increasing, we do not want a human operator watching every uninteresting video feed simultaneously.

We have been working on building a surveillance network including the hardware and the software aspect of it. We have been working towards having a network of around 50 webcams attached to personnel computers which can all share information through a network. This is a network of "smart cameras", meaning that the cameras have a certain degree of computing power at their disposal. The idea is to have the cameras do a certain amount of image processing at their own end. This image processing can involve background subtraction (for a good reference to some approaches read [1] and [2]), detecting blobs in a frame, computation of features like color histograms, sift features [4], Harris Corners, etc. A particular result from sift features matching is shown in figure 1.

For example, some of the challenges faced are being able to describe and finding features that are invariant to scale and orientation. Another interesting issue is what other information can be augmented when stable features cannot be found. These features not only need to be descriptive but they need to be compact as well so that sending them over the network is inexpensive as possible. Also, one cannot just send information over a network of un-calibrated cameras. These cameras have to be calibrated in seven dimensions including time, location and orientation (three dimensions each for location and orientation and one dimension for time).

If we were to transfer all the visual sensor network data (video data) to a central location for processing, the amount of bandwidth required would be enormous. The present implementations of the visual sensor networks tend to be of this format. They aggregate all the information at one location. Power, computational and bandwidth constraints tend to make handling high-volume data a major challenge in sensor networks. That is why, in our implementation we are moving the entire high-data rate processing towards the sensors. Smart cameras that can process live video feed in real time and produce more specific but lower bandwidth consuming data are needed.

Our implementation aims to solve these problems to perform better and more efficient surveillance, tracking and monitoring of interesting objects and events and, remote collaboration. Our goal is to have a general purpose software infrastructure. We have dealt with a broad spectrum of issues starting from the hardware including camera calibration [5] using stereo cameras to software issues from sift features to networking protocols.

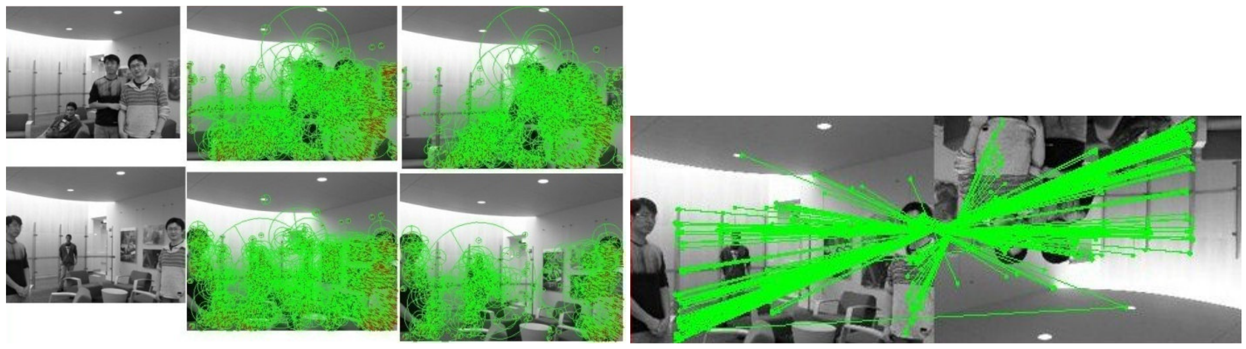


Figure 1

In 1.a the pictures presented are the original frames as captured by two different cameras at two different instances. The next two columns of pictures show the sift features calculated on every pixel in the image with two different thresholds. Fig 1.b shows the matching between the sift features. Besides the occasional incorrect matches most of them are very accurate for frames closely spaced in time from the same camera or for stereo pairs (not in the picture).

Testing SIFT features

In the past, a lot of research has been done on using SIFT features for object matching [3]. Although they work well for stationary objects, they do not when there is a large orientation change. Also in our application the images are of low resolution and SIFT features do not match well in such images. On the other hand features like color histogram are very good for tracking people. We can safely assume that the color histogram of a person will not change much with orientation change, making them invariant to scale and orientation in our application. Also, the color histogram is easier and faster to calculate.

To test the viability of sift features, we tested these descriptors in a network of 12 high resolution cameras (6 mega pixels) to simulate high resolution video stream. These cameras were paired to form six pairs of stereo cameras. We then grabbed frames from stereo pairs and from cameras belonging to different stereo pairs to test feature matching between feature descriptors of the same object viewed from different orientation and scale.

In our tests, we found that sift features that work well for two consecutive frames from the same camera or for the frames from a stereo pair do not work very well across cameras that are far apart in time or in location. No one descriptor works well all the time and hence, in our current implementation we reinforce one with information available from other feature descriptors. An example of this is using color histograms along with the location information.

Current Architecture

The present setup has six pods (camera racks). Each of the pods has four VGA IP cameras. The frame rate is five frames per second. The libraries and the software tools being used are VXL, boost and Qt. SQLite is being used to implement an in-memory database.

The entire framework consists of hardware cameras each of which is managed by its software representation. This software camera grabs a frame from the camera and then registers it to the camera node's main loop. The main loop is where all the logic of the camera node resides. The processor block grabs images provided by the software cam and performs background subtraction and blob detection on them. After blob detection, it runs blob matching which groups blobs into a group called objects and these are blobs that are supposed to belong to the same object in the frame. The processor then stores these blobs and objects in the database for later reference. These blobs detected are then used to satisfy queries made by either other cameras over the network or by the camera itself to satisfy some other query. The network block connects one camera to another. It allows one camera to ask another camera questions about frames, blobs and other cameras. The network block also keeps track of requests (periodic) made by other cameras and then satisfies them as the information from the main block arrives.

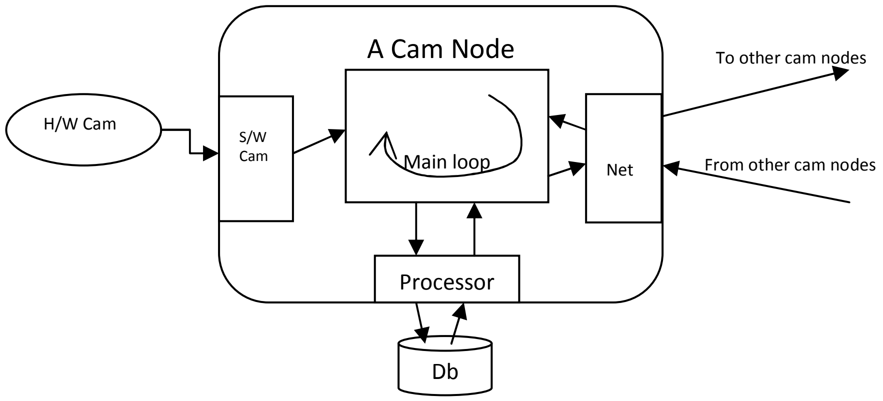


Figure 2

Networking

In the network, only one assumption has been made about the awareness of the network nodes, that is, there always exists a set of known smart cameras in the network. The number of known cameras can be as low as one but at least one must remain up at all times. Other cameras in the network can then just turn up and register themselves at the known cameras. Each camera is assigned a unique id.

Cameras in the network can ask certain questions to another camera. One kind of simple question is where one camera asks another camera for the list of all the other cameras known at that location. A camera can also ask for the current frame from another camera. This can then be used for matching or for camera calibration. One can also ask for the current blobs that another camera can see. When asking for blobs, a requesting camera can ask the other camera to filter some information and not send all the blob details. For example, a blob has information like bounding box, location, color histogram, etc. attached to it. A camera may not need all this information and to keep the bandwidth usage as low as possible, it will not ask for what it does not need.

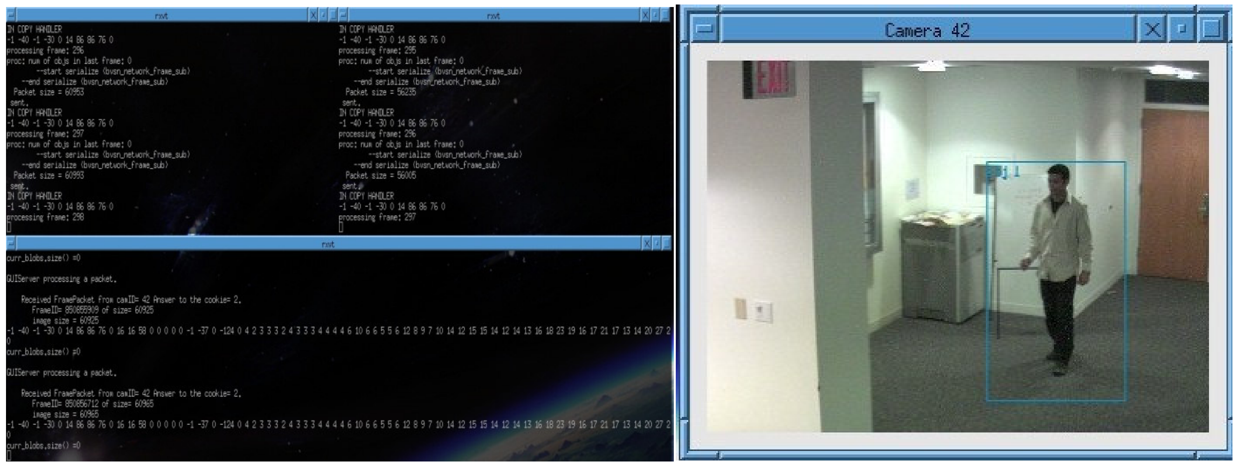
The system allows both persistent and regular queries. The regular queries are satisfied instantaneously. For the persistent query, the requesting camera again has two options. On is a time based query. This query is satisfied after a regular interval of time as specified by the requester. The second type is where the query is satisfied only when there is a change. For example, when a camera sees a new frame, a new object or learns of a new camera in the network. These are more interesting queries and more widely used. Once a camera asks a question, it gets back a cookie that it can use to revoke the question later.

Application writers can write software that talk to this camera network. The camera network does not make any assumptions about the existence of such an application. The application can also ask the same questions that any other camera can ask but it does not need to decipher any information from the camera network. Say, for example a GUI is written. To the camera network, the GUI will be like just another camera. The GUI can connect to the network, ask the known camera for the list of all the other cameras it is aware of and keep sending updates about the known camera list. The GUI can then ask for the current frame and the blobs in the frame from each camera that it is now aware of. The GUI can now display the frames in real time and this GUI is just like a surveillance system monitor.

PUBSUB

The main loop where all the logic of the camera resides has been implemented as a publisher-subscriber model. An example of a publisher is the software camera.

In a pubsub model one can have multiple publishers which are senders of information not programmed to send their message to a specific receiver. Instead, the published messages are characterized into classes, without the knowledge of what subscribers there may be. The publisher subscriber model is an asynchronous messaging model and hence the subscribers only express interest in one or more classes and receive messages that are of interest, without the knowledge of what publishers there are. This allows a great deal of scalability.



```
IN COPY HANDLER
-1 -40 -1 -30 0 14 86 86 76 0
processing frame: 296
proc: num of objs in last frame: 0
        --start serialize (bvsn_network_frame_sub)
    --end serialize (bvsn_network_frame_sub)
  Packet size = 60953
 sent.
IN COPY HANDLER
-1 -40 -1 -30 0 14 86 86 76 0
processing frame: 297
proc: num of objs in last frame: 0
        --start serialize (bvsn_network_frame_sub)
    --end serialize (bvsn_network_frame_sub)
  Packet size = 60993
 sent.
IN COPY HANDLER
-1 -40 -1 -30 0 14 86 86 76 0
processing frame: 298
```

```
IN COPY HANDLER
-1 -40 -1 -30 0 14 86 86 76 0
processing frame: 295
proc: num of objs in last frame: 0
        --start serialize (bvsn_network_frame_sub)
    --end serialize (bvsn_network_frame_sub)
  Packet size = 56235
 sent.
IN COPY HANDLER
-1 -40 -1 -30 0 14 86 86 76 0
processing frame: 296
proc: num of objs in last frame: 0
        --start serialize (bvsn_network_frame_sub)
    --end serialize (bvsn_network_frame_sub)
  Packet size = 56005
 sent.
IN COPY HANDLER
-1 -40 -1 -30 0 14 86 86 76 0
processing frame: 297
```

```
curr_blobs.size() =0

GUIServer processing a packet.

   Received FramePacket from camID= 42 Answer to the cookie= 2.
     FrameID= 850855909 of size= 60925
     image size = 60925
-1 -40 -1 -30 0 14 86 86 76 0 16 16 58 0 0 0 0 0 -1 -37 0 -124 0 4 2 3 3 3 2 4 3 3 3 4 4 4 4 6 10 6 6 5 5 6 12 8 9 7 10 14 12 15 15 14 12 14 13 16 18 23 19 16 17 21 17 13 14 20 27 2
0
curr_blobs.size() =0

GUIServer processing a packet.

   Received FramePacket from camID= 42 Answer to the cookie= 2.
     FrameID= 850856712 of size= 60965
     image size = 60965
-1 -40 -1 -30 0 14 86 86 76 0 16 16 58 0 0 0 0 0 -1 -37 0 -124 0 4 2 3 3 3 2 4 3 3 3 4 4 4 4 6 10 6 6 5 5 6 12 8 9 7 10 14 12 15 15 14 12 14 13 16 18 23 19 16 17 21 17 13 14 20 27 2
0
curr_blobs.size() =0
```

Figure 3

A snapshot of the cameras running (left top half) and a simple GUI based application (left bottom half) that talks to the camera and displays the current frame and the objects detected (right half)

In the pubsub model implemented in the project, there are two main publishers of information. One is the software camera that publishes new frames as and when it receives them from the hardware camera and the other is the processor block. The processor block is both a publisher and a subscriber. It consumes the latest frame, performs background subtraction, blob detection and blob matching to produce objects which it publishes back to the main pubsub block.

To satisfy different kind of requests either from other smart cameras or from the processor block of the same camera, different handlers are registered to the pubsub block. These handlers are subscribers of information that process the messages received from the pubsub block. For example to satisfy persistent queries that ask for the frame to be sent only when a new one is captured the following is what the flow of information will be: The hardware camera captures a frame that is published to the pubsub block by the software camera. The pubsub block then passes this frame to all the handlers that were subscribers of a frame. The handlers then could be of two types. One is a copy handler and it copies the latest frame to a location given to it (say the database) and the other is for satisfying network requests. Both of these handlers can be specialized to do some post processing on the class of information they are subscribed to. The network block for example registers specialized handlers to satisfy request that only require filtered down information on blobs (say blob information without the color histogram information).

Future Work

Our goal is far from achieved. Tracking and identification are interesting research problems that are moving towards practical implementations. We need a system where a user can specify an interesting situation matching certain criteria's using a defined language and then register this as a query in the system. The system should then be able to track this query and when ever satisfied report the even to the system. The user must be able to specify the query without any knowledge of the underlying sensor network or the specification of the sensors or the computational nodes.

These are the issues we eventually need to be able to solve. In this paper we present our current work and our progress towards our final goal. Although we are moving in the right direction a lot more still needs to be achieved. I look forward to the day when our group would have reached that goal.

Conclusion

We have worked on understanding how to use features to best describe objects when detected by our object detection algorithm. We have seen that no one feature works well in all conditions and how some fail across cameras. It is therefore important to use more than one in a way that increases confidence. We have also setup the infrastructure and implemented the framework for future work.

Acknowledgements

I would like to thank my advisor John Jannotti for all his support while I was working with my research team. I would also like to thank Jie Mao and Zhenyuan Zhao with whom I worked in the group. Their inputs always helped me solve complex problems.

References

[1] A Background Model Initialization Algorithm for Video Surveillance - D. Gutchesst, M. TrajkoviCj, E. Cohen-Solalt, D. Lyons, A. K. Jaint

[2] Detecting Objects, Shadows and Ghosts in Video Streams by Exploiting Color and Motion Information R. Cucchiara', C. Grana', M. Piccardi, A. Prati

[3] Large Head Movement Tracking Using SIFT-Based Registration Using SIFT-Based Registration Gangqiang Zhao, Ling Chen, Jie Song, Gencai Chen

[4] Distinctive Image Features from Scale-Invariant Keypoints – David G.Lowe

[5] Distributed Calibration of Smart Cameras – John Jannotti, Jie Mao