Collaborative Video Surveillance

by

Ezzeddine Wessam, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2011

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Ezzeddine Wessam

August 31, 2011

I would like to dedicate this to all my loved ones, whether still with us or not, as a gesture of appreciation for all the care and support they have shown throughout my journey,

Thank You

Acknowledgments

I would like to thank Prof. Vinny Cahill and Dr. Rozenn Dahyot for their advising, supervising and guidance during the entire year. In addition to their attention and time invested in our meetings to put this on the right track.

Special thanks to all my family, here in Dublin and back home, for their love, support and understanding.

Thank you to all my friends for the time they've invested in discussing ideas, sharing thoughts and for their words of encouragement.

As for YOU, I can not thank you enough.

Ezzeddine Wessam

University of Dublin, Trinity College

September 2011

Collaborative Video Surveillance

Ezzeddine Wessam, M.Sc.
University of Dublin, Trinity College, 2011

Supervisor: Cahill Vinny   Co Supervisor: Dahyot Rozenn

Collaborative Video Surveillance is concerned with automating and distributing the detection and tracking of individuals in a surveillance ready environment. It allows camera nodes to detect and track individuals based on an image query created using a mobile phone application. The detection and tracking of the target is accomplished by applying computer vision technology to video streams collected from pre-existing private and/or public cameras. Taking into consideration today's wide availability of smart cameras and powerful technological devices, redesigning conventional surveillance systems provides a better security service that utilizes the wide spread of such technologies. In a network of visual sensor nodes, Collaborative Video Surveillance allows the efficient use of both the available bandwidth and the advanced capabilities of these nodes through automating and distributing the detection and tracking process. This is achieved by processing and analyzing the collected footage locally at the end nodes, rather than being processed in a centralized fashion at the base station. Camera nodes will only return vital information to the base station, thus using a negligible percentage of the available bandwidth when compared to that used by a conventional surveillance system. The utilization of the available bandwidth in such a manner allows the collected footage to be processed at a faster rate, making such a system suitable for real-time applications.

Contents

List of Figures

Chapter 1

Introduction

1.1 Concept

In order to effectively integrate emerging technologies into previously deployed systems that were designed to take advantage of what is currently considered out-of-date technologies, it is required to redesign and rebuild these systems in a way that facilitates and improves the execution of mundane tasks. Collaborative video surveillance allows the deployment of a smart and automated vision tracking system that utilizes pre-installed surveillance equipment and the wide spread of smart phones. The system distinguishes itself from the conventional surveillance systems by the way it automates and distributes the detection and tracking of individuals rather than the common centralized one. This may be accomplished by moving the vision processing functions to a smart camera client node. A smart camera is a camera equipped with sensors/devices that adds processing and networking capabilities to its visual capabilities [5, 7, 10, 18, 19]. Allowing most of the processing to occur locally at the node rather than on a centralized server cuts down on transmission delays, making such a system suitable for real-time applications. Thus cameras can efficiently detect and track an individual target without any human intervention. How will the camera know it's target? The integration of smart mobile phones to the system will allow users of mobile application specifically developed for this purpose, to create an image query that will be used by the client camera nodes through-out the detection process. An image query contains information that defines a certain object and on which future identifications of this object will be based on [1] .In this system, the image query will define the target, and will be used by camera nodes through-out the detection process when searching for the target within their field of view. An image query will be sent to the base station from an android mobile application along with goe-location information. Based on the received location information, the base station will then forward this query to cameras neighboring the target's received location. Upon detection of the target, the camera node will inform the base station of the sighting. It will as well, once the target leaves its field of view, inform neighboring cameras located at the next possible target location to begin a new search process locally or continue an already started tracking process. Therefore, different components, such as the mobile phone and the involved camera nodes, will cooperate in a smart and efficient manner to allow the detection and tracking of a certain suspect. Smart infers to the ability of the mobile phone to capture a photo of the suspect, crop it for better representation of the individual and forward it to a server with other relevant information, and to the ability of the camera nodes to process the captured footage, detect and track the target through the application of complex computer vision techniques. While the term efficient refers to how the detection and tracking process were only cameras located at the targets possible location are involved in the search of the suspect leaving the rest of the cameras idle. It, as well, refers to the efficient use of the available bandwidth by refraining from transmitting video streams back to the base station as in conventional video surveillance applications.

1.2 Motivation

The increasing importance of security in cities and the possibility of using widely deployed CCTV and smart phones to help improve security strongly motivates the design of such a system. Most of the currently deployed surveillance systems are designed to take advantage of technology that is out-of-date and does not take into consideration the technological integration into our mundane lives and the adoption people have exhibited for such devices, smart phones in specific, in the last few years. Since security is normally a top priority for the majority of the nations, making use of modern technologies that are capable of providing a more advanced service is a necessity. The most recent surveys show that around 1.1 million people in The Republic of Ireland own a smart phone, that is around 25 % of the population. These figures have inspired the addition of a mobile application component to the system during the design stage. Further research in the surveillance field, shows that modern smart cameras are becoming more commercially available for less prices on a daily basis. In other words, replacing old cameras that's capabilities are limited to collecting video streams with modern smart ones that are equipped with processing and networking features, will not be as expensive as expected. Integrating the latter technologies together with an aim of providing a better security service describes what the collaborative video surveillance system is. The system will automate and distribute the detection and tracking of individuals in a smart and efficient manner. Thus, utilizing as much of the available technologies as possible in order to improve the deployed surveillance systems.

1.3 Requirements

Collaborative Video Surveillance is a system of large scale that may include a wide variety of features and provide several services. The design, development and testing phases may take several years in order to be completed. However, the time frame assigned for the design, development and testing of this project does not allow for a system of such scale to be built. Instead, a prototype of a smaller scale system grasping the core features of the initial design was developed. Therefore, for this system it is required :

- To design a network that allows the exchange of information between a mobile phone, a base station and several client camera nodes.

- To design and develop a mobile application that allows its user to create an image query. The application should allow the user to capture, edit and send images using the mobile phone.

- To design and develop a smart camera application that is capable of detecting and tracking targets based on the received image query.

- To visually present correlated data received from different client cameras at the base station.

- To efficiently detect and track a target from the field of view of one camera to that of another.

1.4 System Overview

In this section, a scenario describing how the deployment of such a system may be used and benefited from. The scenario will assist in giving the reader an overview of the system components, features and services it can provide. In the city surveillance scenario, citizens who would like to be involved in providing a safer place to live will only be required to download the mobile application to their phones and to use it when necessary. The users will then use this application to create an image query of the suspect when required. The transmitted image queries will be received at the base station, which will normally be monitored by someone who has a basic understanding of the system and the underlying components. He/She may as well assist by confirming possible sightings received from the camera nodes, thus removing any false positives that may result during the detection process. In regards to the camera nodes, they're usually modern surveillance cameras that are deployed in the city and that are capable of executing certain computer vision techniques and equipped with networking capabilities.

Since the latter is it a bit complex to build and requires several permits before

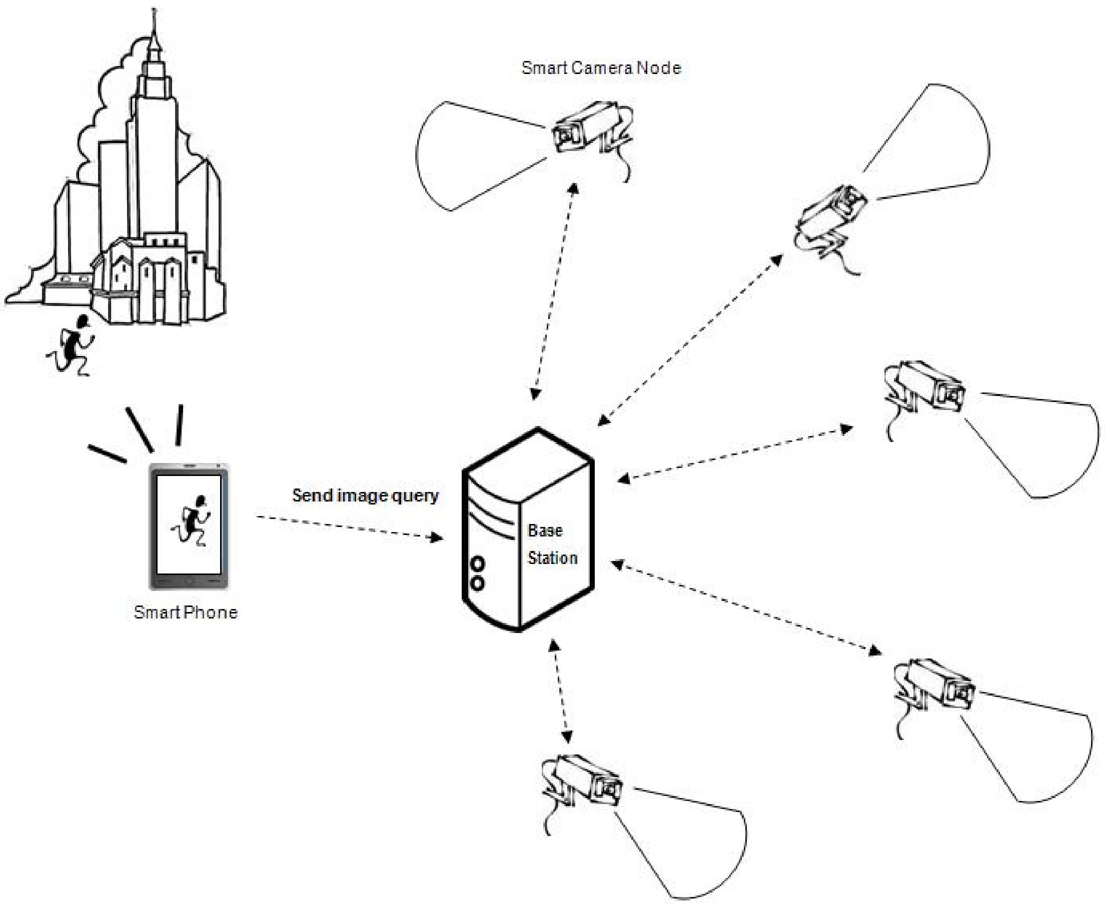


Figure 1.1: System Diagram.

implementing, it will be impossible to complete such a project in the time frame allocated for this Masters course. Therefore, a prototype built inside one of the university's labs will be used to demonstrate the operability of the system and the features it's expected to provide. In the lab, two local area networks will be setup, one connecting the mobile phone wirelessly to the server and the other connecting the server to the camera nodes. The connection between the base station and the client camera nodes can be either wired or wirelessly, depending on the kind of equipment used. The visual capabilities of the camera nodes will be simulated by storing footage previously collected by CCTV cameras deployed around the city. Streaming in an offline manner as the one described, was preferred over connecting webcams to PCs and analyzing real-time streams. This is due to the fact that the footage collected by the CCTV cameras is closer representation of the kind of video the system will have to deal with when deployed in a real world. The similarities are mostly related to the brightness levels, the size of the target and the angles they might be view at. A photo will be taken using a smart phone, cropped and sent to the server and then distributed to the camera nodes. A map on the server showing the geographical locations of all connected camera nodes will indicate the location of the target upon detection and will allow the visual tracking of the suspect on the map.

1.5 Achievements

The prototype designed and developed for this system was capable of detecting and tracking individuals from the field of view of one camera to that of another. The prototype was designed to be tested offline. Therefore, there was no real time streaming. The camera nodes would analyze footage that has been previously gathered by CCTV cameras and has been stored locally. The detection and tracking process is based on an image query created using an android mobile phone application. Once created, the image query is transmitted to the base station, and then from there forwarded to the client camera nodes. In addition, the base station provides visual feedback to the user based on information received from client camera nodes through-out the detection and tracking process. The feedback provided indicates the target's current location, as well as the direction he/she might be moving in.

1.6 Dissertation Structure

Chapter 2 presents the state of the art of this project, which mainly focuses on the previous work that has been done in fields related to distributed surveillance and vision based tracking. It may as well include other technologies that may be associated with the collaborative video surveillance system.

The Design Chapter (chapter 3) illustrates the design decisions that have helped sculpt this system, including software, languages, libraries, algorithms, topologies and many more.

The implementation Chapter (chapter 4) provides a more in depth discussion on the algorithms implemented, through the use of examples.

This is followed by the Evaluation Chapter (chapter 5), that analyses the performance of the overall system as well as the individual components and algorithms used.

Chapter 6 concludes, including a section that presents some of the future work that can be added to the developed system, thus improving it's performance and integrating extra features that it may lack.

Chapter 2

State of The Art

This chapter discusses previous research that is related directly or indirectly to the design and implementation stages of this system. The methods, algorithms, technologies and architectures discussed below are part of the system's design. However, not all the mentioned technologies will be used in the development of this prototype as their use is considered invalid for a system of such a scale.

2.1 Introduction

Collaborative Video Surveillance describes the concept of distributed surveillance in which monitoring is not restricted to the field of view of a limited number of individual cameras that operate independently but rather dependent on distributed smart cameras that operate cooperatively to provide a monitoring/tracking service. Smart cameras are camera nodes equipped with video sensing, processing and communication capabilities, they are also known as visual sensor nodes (VSNs) [5, 7, 10, 18, 19]. The tracking service for which such cameras are deployed involves the detection of objects of interest and the tracking of such objects from the field of view (FoV) of one camera to another. Such a system makes use of pre-installed private and public surveillance cameras without compromising the privacy of the environment that are being monitored by these cameras. In other words, the collected video streams are processed and analyzed by machines, with no human intervention. Therefore, the collected footage is not monitored by people directly, thus human privacy is conserved. The system reacts to image queries received by the server from a mobile application running on a smart phone. The queries are routed to cameras located at addresses relevant to the targets candidate locations. Cameras that receive the queries analyze both recorded and real-time footage in search of the described suspect. The search process is implemented using complex image processing techniques, mainly related to pattern recognition and comparing images. Cameras within the network will collaborate throughout the tracking process, in order to provide a more accurate and efficient tracking process. This is due to the fact that the search process will only be triggered at camera nodes that are around the current targets location. The information passing between the camera nodes may as well limit the number of false positives identified, however, this will require some human intervention who will validate true positives upon detection. Efficiency may as well be improved by sharing information in a decentralized manner, enforcing spatial and temporal collaboration and the use of data aggregation techniques. The use of such a system is not solely restricted to tracking of suspect individuals but may serve various other purposes, for instance, tracking lecturers within a lecture hall (in such a scenario cameras may have overlapping fields of view, which may not be true in video surveillance). A very similar system has recently been deployed by the Chicago City government, providing security personnel with an eagle eye's view of all on-going events on the city's streets [4].

The goal of this paper is twofold. First, it will discuss the middleware layer that underpins the overall system, allowing the flow of information between it's various components. The middleware section discusses some of the routing algorithms that have been used to design such system. The section will as well focus on other aspects such as node collaboration techniques that will govern how the communication between among the client camera nodes and between the base station as well. As for the computer vision section, it is mostly related the design of the computer vision component of the client camera node. The section will discuss computer vision techniques that may be used for the detection and tracking of the target based on image queries. Section 2.2 introduces the middleware part of this project, discussing the issues of network topology Section 2.2.1, routing algorithms Section 2.2.2, data fusion Section 2.2.3, node collaboration Section 2.2.4, and scalability Section 2.2.5. Section 2.3, discusses the vision techniques used for image querying Section 2.3.1, object detection Section 2.3.2, object tracking Section 2.3.3, and the occlusion problem Section 2.3.4. Section 2.4 describes this project's methodology, and finally section 2.5 concludes.

2.2 Middleware

The integration of visual sensor nodes into a distributed and collaborative network is normally supported by a well defined middleware that abstracts the physical devices into a logical model, providing a set of networking and communication services [22]. Such networks are known as VSN. Taking into consideration data fusion and node collaboration techniques, will aid in efficiently executing such services . The section will focus on designs and algorithms that will define the middleware layer underpinning the vision applications running on higher layers in the client nodes. Technologies that are relevant to VSNs deployed for surveillance purposes are only taken into consideration.

2.2.1 Network Topology

Smart cameras or visual sensor nodes are a type of sensing device deployed for the purpose of collecting visual information about their surrounding environment. Their main task involves monitoring the area covered by their FoV. Therefore, nodes need to be linked according to some predefined network topology to allow efficient communication to take place within the network, which also takes into consideration the geographical locations of these nodes and the covered FoVs.

Issues arise as to whether the system should have a centralized or distributed architecture. The system should allow multiple cameras to exchange information in a manner that will aid in tracking a target in real-time. Information exchange can either be done through a centralized node or in a distributed fashion in which nodes are able to exchange information directly. In the latter case, implementing a distributed architecture imposes several difficulties and challenges, especially in the case where the system's components are visual sensor nodes. For example, the lack of a central station requires that nodes have some knowledge of other network nodes for routing purposes, as well as synchronization difficulties and extensibility. In addition to these problems, VSNs suffer from issues related to determining the FoVs of each camera node, which adds to the complexity of both the design and implementation process. This is due to the fact that the range covered by visual sensor nodes depends on the direction in which the node's lens is pointed, as well as the type of lens(wide angle lens, etc.). Unlike other types of sensor nodes, the range covered by visual sensor nodes is not circular but rather diverges in one direction. Taking these issues into consideration, developing a distributed system of wireless sensor nodes may be considered quite ambitious, but the expected improvements in network performance, reduced energy consumption due to the decrease in the number of transmissions and efficient routing, resolving the single point of failure and bottleneck issues, and several other benefits will make it worthwhile. For example, a distributed system is tolerant to the implications of bottlenecks and central points of failure from which centralized systems constantly suffer, not to mention the latency that could be involved when such a system is to be used for a real-life application, such as real-time tracking. Additional latency costs may be a result of the large number of transmitted packets in camera sensor networks [19]. Round trip delays will have an impact on decision making. The latency centralized systems introduce when constantly routing packets back to the base station may cause the system output to be somehow irrelevant, since decisions may be based on stale data due to transmission time delays. Therefore the system may not serve the purpose for which it was initially developed. In relation to data communication, centralized surveillance systems transmit the data gathered at the sensor nodes directly to the base station, creating an immense amount of traffic due to the large size of collected footage, leading to an increase in cost, as well as causing network congestion and data packet loss. Communication can be more than 100 times more expensive in terms of energy consumption than processing [19]. Therefore, processing data locally and compressing it prior to transmission, as well as only transmitting data to relevant nodes through a minimum number of hops has a positive impact on bandwidth requirements, reduces heat dissipation and is considered as a more efficient technique. Such advantages make a distributed system more suitable for video surveillance where the size of the data collected by the visual nodes is quite large.

Within a distributed architecture, different network topologies may be considered. Charfi et al [6], discuss how grouping visual sensor nodes into clusters may enhance the performance of the network. Forming clusters within a network, often referred to as a hierarchical topology, divides the network into smaller groups of nodes. Each group or cluster is assigned an aggregation node. The aggregation node is responsible for routing traffic to and from the edge nodes of its cluster, as well as filtering and fusing the received data [7, 14]. In addition, the node connects to other aggregation nodes and to the BS. Other types of topologies include the mesh or decentralized topology, in which all nodes are considered as equals and are able to communicate directly with each other [32]. Another possibility would be a hybrid topology, which normally are a mixture of the above mentioned topologies. Mesh networks require that all the nodes in a network acquire information related to the other nodes with which it might communicate with. Therefore, extensibility might be considered quite a complex and costly option. In addition, setting up a network for a mesh topology is a complicated step and may not be easily automated.

A distributed network adopting a hierarchical topology is to be considered for this system. Visual sensor nodes are grouped into clusters according to the geographical locations of their FoVs and are assigned an aggregation node. Aggregation nodes are aware of the network and will be able to forward data to relevant nodes efficiently through applying data fusion techniques.

2.2.2 Routing Algorithms

A routing algorithm is a methodology that allows the communication of network traffic along a path determined by the defined methods. Routing algorithms in sensor networks tend to be different than these used in LANs, due to the limited processing and power capabilities of the nodes,as well as the purposes and environments they're deployed for. For sensor networks, routing algorithms are designed in a way that takes into consideration the limited resources available, especially the limited power supply. However, since camera sensor nodes in the VSN to be built for this project have no power restrictions, the considered routing algorithms are not evaluated with respect to their efficient power usage. Liu et al. [14], discusses two of the most popular hierarchical routing protocols LEACH and PEGASIS, as well as a third proposed routing protocol combining the two. LEACH (Low Energy Adaptive Cluster Hierarchy) divides the network into clusters and assigns each a cluster head. It uses the cluster heads to communicate traffic between the BS and the network nodes. As for PEGASIS(Power Efficient Gathering in Sensor Information System) nodes connect to their closest neighbours until a link is created connecting the end nodes to the BS. Master nodes are the only nodes allowed to communicate directly with the BS. Master nodes are re-elected on a regular basis. However, since the PEGASIS routing protocol tends to form a chain structure among the nodes, the LEACH protocol is considered to be more appropriate for the hierarchical cluster-based topology adopted for this project. Unfortunately, acquiring source code for such protocols has proven, from past experience, to be quite difficult. Therefore, a more widely used protocol such as the Uniform Datagram Protocol(UDP) may be used. In

UDP, nodes may transmit packets, known as datagrams, onto the network without the need to establish a transmission channel with the destination node [23]. The UDP protocol was not specifically designed to be used in VSNs, however, since the camera nodes used for this project are already connected to a backbone network, adopting a protocol such as UDP is considered to be effective and efficient.

2.2.3 Data Fusion

Data fusion is a mechanism that reduces the amount of data transmitted in a network by employing local processing at nodes within the network. Data received at a forwarding node is combined with local data prior to being forwarded. The process includes the use of data redundancy checks and some aggregation algorithms. Charfi et al. [6], explain how data fusion can be used in VSNs by referring to various techniques such as visual data filtering and visual data coding. The latter is a process that reduces the size of transmitted data by encoding packets locally prior to transmission. While the former, visual filtering, would include methods such as converting from 2D to 1D, lowering image resolution, checking for redundant data and filtering images that do not indicate any change. Wu and Chen [31], have implemented a different technique that works in conjunction with an already existing image processing algorithm. The technique transmits the background and target images separately making use of the background subtraction technique used for detecting moving objects. The background image is transmitted only once, when the nodes are initially set up. Afterwards, only the subtracted image containing the detected object is transmitted after being fused with spatial and location information relevant to the node and the detected object. The receiving node, then reconstructs the image by using both the background image and the received data. This technique has been proven quite effective especially in the case were the FoVs of the visual sensor nodes are overlapping. The effectiveness of a technique is normally measured by comparing the technique's computational costs with the transmission costs that it's meant to reduce. Therefore, it is important to note that decoding the segments at the receiving nodes should be computationally cheap otherwise such a process would be considered infeasible and cannot operate in conjunction with other data fusion techniques. Consequently, Wu and Chen's [31] technique is considered more efficient and effective than that described by Charfi et al. [6], due to its implementation simplicity and cheap computational costs. That being said, both techniques can work in conjunction with each other, if necessary, since their implementations do not overlap.

2.2.4 Node Collaboration

At the heart of a distributed system, lies the cooperation between the different nodes and the knowledge that each has of the other. Node collaboration is the process by which nodes store and process data collectively rather than autonomously. Whether the collected or produced data should be kept local or shared depends on the type of the data and its relevance to the task at hand. For this project, it is more important to focus on the sharing of data produced from processing information collected by the visual sensor nodes. Rinner et al. [18], mentions how smart cameras may be classified into three different categories: single, distributed and pervasive smart cameras. In single smart cameras data processing occurs locally without taking into consideration the results produced by other nodes within the network. This evolved into a much smarter system, defined in the Distributed Smart Cameras (DSC) and Pervasive Smart Cameras (PSC) categories where the collected data is processed in a distributed and collaborative manner. PSCs take DSCs a step further by integrating adaptivity and autonomy with the aim of providing a service-oriented network that adapts to various changes. DSCs tackle the task at hand from different angles by taking the bigger picture into consideration. This may be accomplished by taking the field of views of several cameras and acting upon the inferred knowledge from these views rather than making decisions based on the FoV of an individual cameras. In other words, implementing collaboration between the nodes belonging this network. For this, a network of DSCs it considered an improvement to a network of single smart cameras. Agents - tasks that can be triggered as distributed applications - may be used to assist in the distributed processing. This may accomplished due to the dynamic nature of agents, allowing tasks to be executed when required depending on specific state of the whole system. PSCs extends DSCs by adding extra features such as the addition and removal of nodes without the need to manually configure the network as well as the level of adaptivity with which the nodes operate. DSCs and PSCs rely on calibrated image data in the spatial and temporal domain. Spatial and temporal collaboration is a technique which allows nodes to identify their locations with respect to the other nodes in the network and to synchronize to the same time. The technique is further discussed in [13], where a map of the FoVs belonging to camera nodes within a network is created based on the available spatial and temporal data. Such techniques are used for the implementation of 'application-specific' and 'application-independent' programs. Application-specific features are these that are related to a certain higher level task, while application-independent are related to lower level task that are more related to the platform than to the system's main goals. Spatial and temporal calibration allows source nodes to identify destination nodes for their data packets by defining the location of nodes with respect to each other. To aid in the development of such systems, mobile agents may be used.

A mobile agent is a software that resides on camera nodes for representation and management of the node's activities [16]. Agents have partial knowledge of the state of the system, by combining locally generated information with that generated by other mobile agents within the network. Therefore, decisions made by a mobile agent are normally dependent on the limited information offered, thus promoting distribution among the camera agents. Their advantages range from scalability to fault-tolerance, especially when compared to centralized systems. Mobile agents operate in coherence with the spatial and temporal calibration technique mentioned previously since it relies on the spatial information produced in the calibration phase. Rinner et al. [17], represent different types of services required for implementing a distributed system such as deployment, operational, networking and application-specific services. From these, mobile agents are normally used for operational services. A case study related to tracking individuals using multiple cameras in a decentralized system of DSCs, where agents represent an object tracking task that was created on demand. The agent will then 'move' from one camera to another, following the target's movement. Thus the agent continues the task it started on the first camera, on the new camera at which it is currently located. In other words, the agent processes the captured footage only on cameras that have the target within their FOV. The code mobility concept is further discussed in [19], where a multi-camera system uses agents for following targets among cameras. The experiment highlights the importance of having very well defined spatial and temporal data. For the implementation of VSN, mobile agents are known to be quite useful given their support for scalability and fault-tolerance.

Another method that takes advantage of the distributed nature of the network to achieve efficient processing, analysis and sharing of the collected data is the use of a Service Oriented Architecture [13]. In a Service Oriented Architecture a node is capable of providing a set of services. Such services may not operate all at once,therefore enabling and disabling of a service depends on other attributes. Nodes will enable services as they are required while keeping the rest of the services disabled. The dynamic disabling and enabling of services is made possible by modeling the system into three major layers:

- application
- service
- middleware layers

Even though such a technique efficiently uses the available processing resources, it consumes space by storing services that might rarely be used. Available space is crucial in distributed video systems, since nodes have limited resources and the gathered footage takes up a significant amount of space. Besides that, adding new services to the system that were not part of the initial design will require upgrading of the entire system. The dynamicity and scalability of mobile agents makes it suitable for a distributed system. Unlike other techniques, mobile agents allow the design of more robust and decentralized systems.

2.3 Computer Vision

Computer vision allows information extraction from an image for inspection, analysis and control purposes [21]. These images can be stills as well as video frames and are either processed as greyscales, colour or multi-spectrals. The manipulation of images is normally achieved through a combination of image processing algorithms that are applied consecutively depending on the task at hand. Algorithms range from basic image processing techniques such as digitization and smoothing to more complex ones such as object recognition and motion analysis. Applying these techniques could be quite difficult if not provided with the right tools. However, with the aid of high-level computer languages such as C/C++ and using a powerful library such as OpenCV [27], applying such algorithms can be done in an efficient and reliable manner producing very accurate results.

2.3.1 Image Querying Techniques

Image querying is the process of identifying objects that are relevant to a query using a semantic based representation of the expected result [3]. Image querying is normally divided into a semantic-based and content-based approaches. The semantic-based approach represents the image using metadata and queries the image database for similar context. As for the content-based approach, the query is executed by matching low-level features such as texture, edges, color histograms and many more. It is normally referred to as Content-Based Image Retrieval (CIBR). Users will create an image query by taking a photo using their mobile phones. This photo is geo- tagged and forwarded to the BS. Geo-tagging the photo allows to forward the created image query to camera nodes neighbouring the location in which the photo was tagged. The BS processes the image prior to forwarding it, isolating the target object from the background. Then, the receiving nodes will analyze the collected frames as well as previously gathered footage in search of the queried target.

2.3.2 Object Detection

Object detection allows for the identification of semantic objects using computer vision techniques. It may be a very complex process unless provided with the right tools and data sets that define how the object may appear from different angles and under different lighting conditions. Diehl et al. [8], discusses how object detection is applied in the real-time classification and detection of moving objects. The technique presented mainly focuses on the use of classifiers. Using classifiers for object detection gives context definitions under which the detected objects may fall. Therefore, object recognition is restricted by the quality and the quantity of the original specified context. Techniques attempting to automate the learning phase have been developed, but they are prone to error when not guided by humans. The learning phase allows the application to gather information that will be used to improve the detection of true positives and limiting that of false positives. However, semi-automation is quite useful in which both man and machine may collaborate in offering a continuous learning service. This could be done by requesting the user to classify an unidentified object that has been detected by the system.

Wolf et al. [30], discusses the early developments of a DSC system that may be used for detecting and tracking individuals, as well as activity recognition. The classification of human activities is not of interest to us in this project, but the Hidden Markov

Model used in the activity recognition process may be used for object detection to reduce the number of false positives a system might produce during the object detection process. The paper presents the process of detecting humans from the captured footage using low-level image processing methods which consists of a series of four different algorithms applied consecutively to detect and identify different parts of a human body. Region extraction, contour following, ellipse fitting and graph matching are the four algorithms used for identifying humans, especially body part detection. Region extraction allows the separation of the foreground and background in a captured frame, as well as the detection of the body part's skin area using the luminance and chrominance(YUV) colour model. This is succeeded by the contour following technique, which groups the identified pixels into contours. Then, ellipse fitting corrects deformations produced during the image processing caused by clothing, objects in the frame, or due to occlusion. Finally, graph matching, classifies the parts identified during the previous steps with respect to predefined data. The system was tested and performed as expected, even though real-time response varied in accordance with the complexity of the applied image processing techniques.

The classification technique presents a method that allows to separate objects into groups according to a certain classifier or feature. Therefore, it could be considered quite inefficient when used for tracking human individuals. As for the latter, the series of algorithms used are considered quite effective if used for human detection, tracking and activity recognition, however this is not the case for this project. Therefore, simple techniques such as template matching are found to be the most appropriate due to their simplicity and cheap computational costs.

2.3.3 Object Tracking

Tracking has been the focus of the majority of computer vision researchers for the past decade. As a result, a variety of techniques have been developed for tracking objects located within the FoV of one camera or moving from the FoV of one camera to another, also known as multiple-camera tracking. Some of the most common algorithms used are the mean shift, Kalman filter and particle filtering algorithms. The most effective way of implementing tracking is through a combination of these algorithms. Rinner Wolf [19] discuss how effective combining the spatial masking of histograms with an isotropic kernel followed by mean shifting can be in tracking moving objects. He also mentions contour following techniques for tracking people without the need for a 3D model and Kalman filtering for predicting trajectories to deal with occlusion issues. As discussed in [12, 29], tracking multiple objects in multiple cameras may be a very challenging task. It requires spatial and temporal calibration of all camera nodes within the network, the locations of their FoVs and whether they overlap or not. Such issues may be overcome by determining the FoV of each camera within the network in order to simplify the object hand off process. Object hand off is the movement of an object from the FoV of one camera into that of another. Furthermore, an object entering the FOV of one camera may not necessarily be heading into the FOV of the following camera. Therefore, spatial calibration assisted by object labeling will reduce the number of false positives produced. Therefore, objects detected in one camera must be assigned labels and such labels must be shared with other cameras in the network in order to allow the successful tracking of the target object in an area covered by multiple cameras. The establishment of some correspondence between the camera nodes in the network must precede any object detection and labeling activities. Correspondence may be established by identifying the edges of each FoV and finding which FoVs are closer to each other. Overlapping FoVs must be taken into consideration as well. Such a technique may be very useful in the case of having multiple cameras covering the same area. As for establishing correspondence between non overlapping cameras, this is done using spatial calibration which takes may also take into consideration geographical locations.

Others [2], have focused on tracking individuals in a large-scale smart camera network using a signature created from classifying objects according to descriptors based on a Difference of Gaussian detector. A signature specific for each object is created at the identified node and shared with all camera nodes within the network. Receiving nodes will store the signature in a buffer and compare it with all objects within the FoV of the camera. A predefined threshold allows for the removal of noise and for the recreation of a more robust description of the previously-defined object. Even though the methods described take into consideration the temporal synchronization of the camera nodes using the Network Time Protocol (NTP), they, fail to mention any sort of spatial calibration. The object signature is transmitted using broadcasts to inter and intra group members - members belonging to same or different clusters respectively. Such methods may be quite inefficient due to increase caused on the network load and on local processing at the nodes in search of a matching object, especially in large-scale camera networks. On the other hand, the reacquisition and tracking of objects has proven to work as expected in given experiments.

2.3.4 Occlusion

The occlusion problem may be summarized as the case when an object is in the FoV of a camera but cannot be observed in the acquired footage due to the presence of an obstacle between the visual node and the target object [9]. Occlusion is a very common issue, especially when visual sensor nodes are deployed in public areas that are often crowded by people. In this case, an individual target may not be detected by the monitoring camera due to the presence of other objects between the target and the camera, mainly another person. Consequently, occlusion is an issue that will not be dealt with in the work applied for this project.

2.4 State of The Art Conclusion

The above mentioned techniques may be utilized in a way to provide a collaborative video surveillance system that is capable of monitoring and tracking suspects in an efficient and smart manner. This could be achieved by implementing a robust and dynamic visions layer supported by a transparent, extensible and scalable middleware layer. The higher level vision layer may benefit from the use of a combination of the described querying, object detection and tracking techniques. While the middleware layer is designed to be distributed and fault-tolerant, decreasing the possibility of network failures and packet losses.

Chapter 3

Design

This chapter discusses the design decisions that have been made in relation to the system components, the methods these components use to communicate among them and the algorithms that will run their individual engines. Each of these components, the mobile application, the base station and the client camera node, will be discussed individually and as a system, providing an in depth explanation on each of their elements. In addition, a section on the system's architecture will allow for a better understanding of the overall design and layout of the whole system. Finally, a brief summary on the Iterative and Incremental Development model, as it was the adopted development model.

3.1 System Architecture

This section gives an overview of the system and it's individual components. It as well lays out the links that bridge these components together.

3.1.1 System Overview

The system is comprised of three interconnected components. The links bonding these components together represent the type of communication used to allow the flow of information within the system. The links could either be temporary or permanent depending on the components it connects, as each component provides a system feature. For instance, the mobile application connects temporarily to the base station when the user wishes to send an image query. It is impossible and unnecessary to gather any sort of knowledge on the transmitting mobile phone a prior to transmission, as it is neither beneficial nor required by this system. However, when referring to the client camera nodes, it is essential to gather a prior knowledge regarding each camera node. This is usually done when a connection is established between that connecting node and the base station. The information allows the base station to assign a unique ID to this camera node that will map to the camera's provided geographical location. Such information will be used by the base station when attempting to visually represent the target during tracking.

3.1.2 System Components

The system is comprised of three components, a mobile phone application, a base station and a client camera node. The mobile phone application is designed to be used an image query creating tool. Once created, the query will be transmitted to the base station and from there forwarded to relevant client nodes. The application allows its user to capture a photo using the device's built in camera and crop the image using the provided image editing tools prior to sending it to the base station. Cropping the image specifies exactly where the target is in the captured photo, creating a more accurate image query. As for the base station, it's main aim is to bridge the mobile and client node components together to allow the image query to be used as a template by the client camera node. It as well is responsible for visually presenting the location of the target, upon detection, to the user. The visual representation includes an on map indicator, an arrow pointing in the estimated direction of movement, as well as snapshots of the identified target. The addition of the snapshots feature allow the user to identify false positives that have been wrongly detected by the client camera nodes. As for the client camera node, it is responsible for the detection and tracking of the target defined in the received image query. Upon detection the client node transmits regular updates back to the base station.

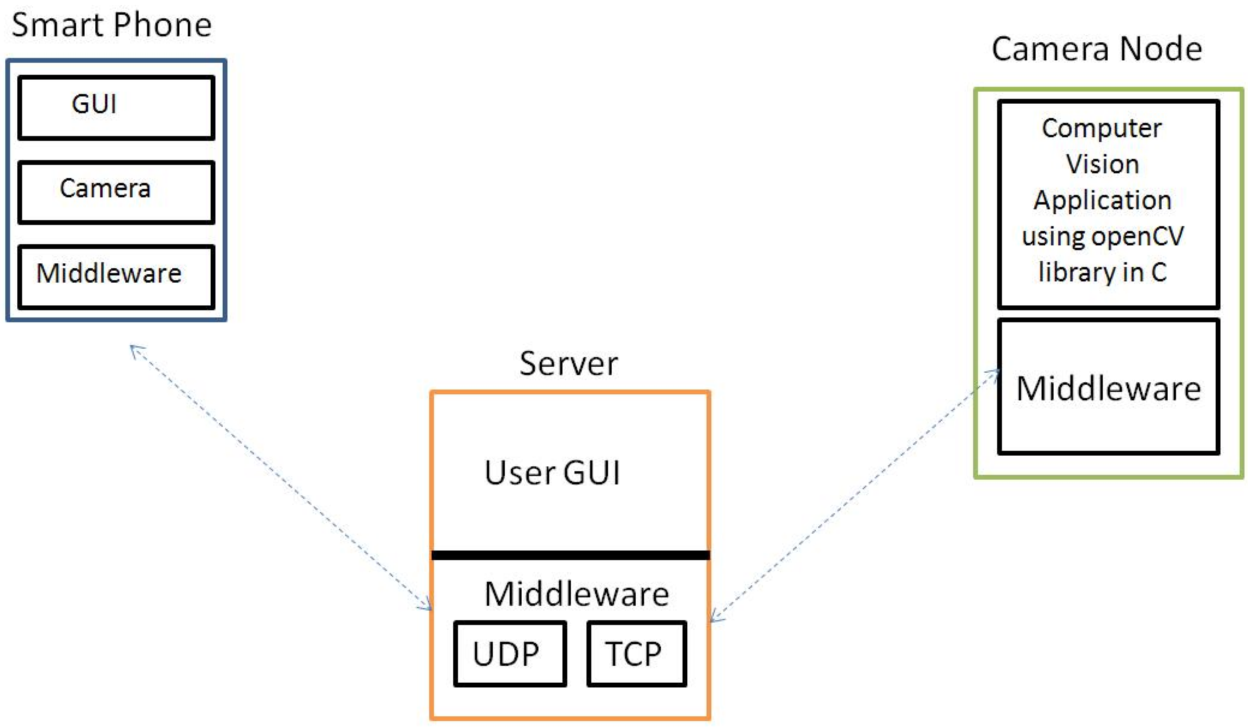


Figure 3.1: Components Diagram.

3.2 Mobile Application

A Mobile application is a piece of software that runs on smart phones, usually presenting the user with a friendly graphical user interface that allows him/her to make use of the underlying hardware[15]. In this system, the mobile application is the first step taken towards the detection and tracking of a target. It provides the user with image capturing and editing tools to complete this task. In this section, the motivation behind the choice of the Android OS and how the image query is created and transmitted to the user will be discussed.

3.2.1 Mobile Platforms

When discussing the platform upon which it was intended to create the mobile application for, there were several very well developed and popular mobile OS. Some of these are:

- Android
- iOS (the platform for Apple i devices)
- Symbian
- Web OS
- Windows Phone 7

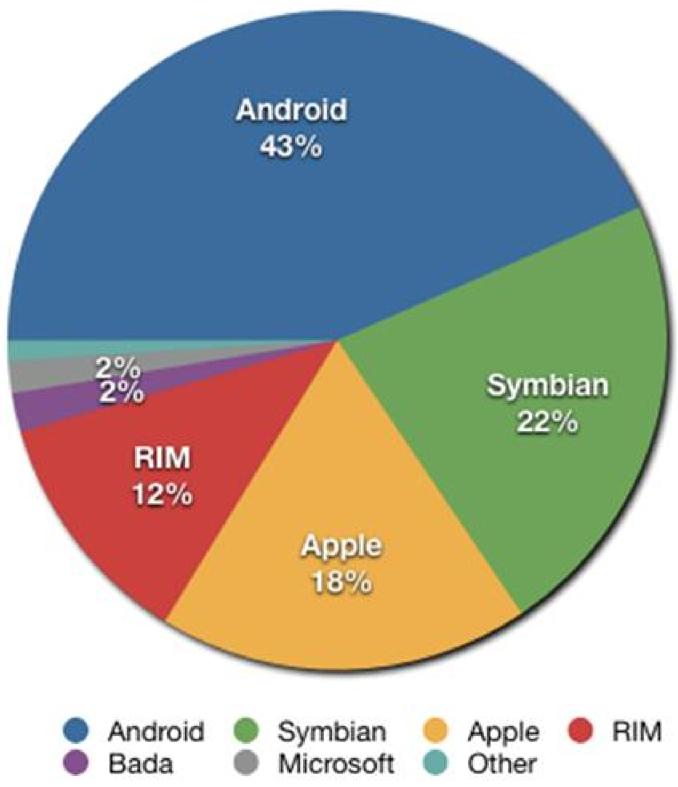


Figure 3.2: Mobile OS Market Share.

Looking into the platforms, it was discovered that all platforms provided similar features which would be of use to the system, such as access to GPS, networking and advanced graphics. The next was narrowing down the possible platforms. Limited availability of either Web OS and Windows 7 phones, removed the latter off the list. Taking into consideration that news that have been out in the past few weeks, not developing for Web OS was a good decision. Symbian was ruled out for similar reasons, the unpopularity of the Nokia devices in the smart phone market the devices available were lower end devices. In addition, recent decisions taken by Nokia to adopt the Windows 7 Mobile to be used for its phone. This means that there's a no market for applications developed for such platforms. Which leads to less people buying Nokia phones, thus less people downloading and using the application if it were to be developed for such a platform. Therefore, the two final platforms left would be with Android and iOS. The Android OS was the platform of choice due to the following:

1. Developing using familiar languages such Java and XML, languages used for Android development, was preferred over the use of Objective C.

2. The SDK for Android is a cross platform with versions for Windows, Mac OS X and Linux. While the SDK for iOS is designed only for Mac OS X.

3. The SDK for Android is free of charge and easily allows a user to test an application on a number of different devices simultaneously. The iOS SDK has a free version which does not allow the user to test an application on a device, only the 99 SDK will allow for device testing which is more limited in the number of devices which can be used for testing than the Android SDK.

4. Another factor that was not as important as the ones listed above but which was considered when deciding, was the rapid growth which Android has been experiencing which resulted in the platform becoming the largest selling mobile platform worldwide by the end of 2010. There's around 500,000 Android devices that are

being activated on a daily basis around the world.

5. Previous experience with Android development was a factor as well.

3.2.2 Android OS

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. It is based on the Linux Kernel [25]. It features:

- An application framework for components replacement and reuse
- A Dalvik virtual machine optimized for mobile devices
- An Integrated browser
- Optimized graphics powered by a custom 2D graphics library
- SQLite for structured data storage
- Media support for common audio, video, and still image formats
- GSM Telephony
- Bluetooth, EDGE, 3G, and WiFi
- Camera, GPS, compass, and accelerometer
- Rich development environment including a device emulator, tools for debugging, memory and performance profiling, and a plug in for the Eclipse IDE

The diagram in Figure 3.3 gives a detailed explanation of the major components compromising the Android operating system.

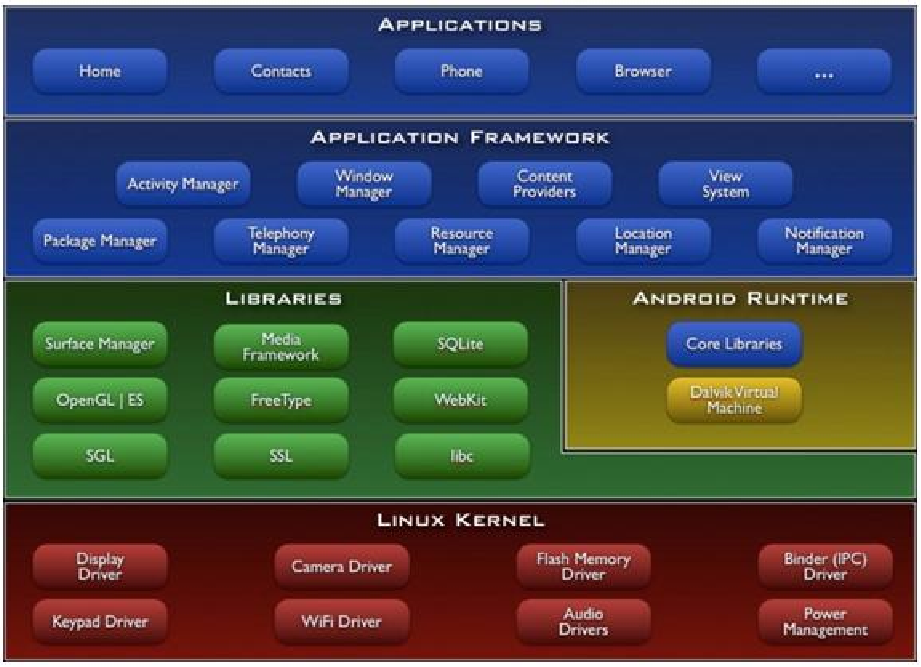


Figure 3.3: Android System Diagram.

3.2.3 Android SDK

The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language. By providing an open development platform, Android offers developers the ability to build extremely rich and innovative applications. Developers are free to take advantage of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, and much, much more. Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user.

3.2.4 Android Components

Android applications are designed to be made of a set of predefined application components. Each of these components provides different access methods for the system to enter an application. Components exist as entities on their own and do not necessarily form a point of interaction with the user, as they may depend on one another. These components can be defined as a unique building block that helps scope an application's overall behavior.

There are four different application components that a developer can use while building an android application. Each type serves a specific purpose and has a unique life cycle that sets how the component is created and destroyed. These four components are Activities, Services, Content Providers and Broadcast Receivers. Below you will find a more in depth explanation of the first two components, Activities and Services, as they are the two components used while developing the mobile application for this system.

- **Services:** A service is a component that performs work for remote processes or performs long-running operations. It is mainly distinguished from activities by how it runs, as it executes in the background and does not provide any kind of user interface. For example, a service may play music in the background while the user is interacting with a different application. A service may as be used by several activities or applications that bind to it while its running to make use of the methods it has available. It is implemented as a sub class of Service.

- **Activities:** An activity represents a single screen with a user interface. Although activities collaborate together to form a cohesive user experience, each one is independent of the other. Each user interface will provide the user with certain features and methods he/she can use. They appear to the user as a single application, however, it is formed of several activities and other components that interact with each other to provide such an experience.

- The mobile application developed for this system allows the user to capture a photo, edit and send it to a server. The photo capturing and editing features were implemented using the activity component. As for the networking aspect of the application, a service component was extended to allow the segmentation and transmission of the edited photo.

3.2.5 Mobile Application Components

This section discusses the mobile application that has been developed as part of the collaborative video surveillance system. The application was developed for the Android mobile OS using the android SDK plug in for the Eclipse IDE.

Mobile application overview

Since the android SDK is based on both Java and XML languages, the mobile application is separated into two parts, one that is in XML, mostly related to the GUI and the other is in Java, and usually relates to all the developed methodologies. Components, if activities, may be a result of a combination of code written in both Java and XML. However, this is not the case if the class extends a Service. The main application components include:

- A class for accessing the built-in camera when capturing a photo.
- A class for handling touch events and android graphics.
- A class implementing the middleware layer, which deals with incoming and outgoing traffic.

Application Components

The application is divided into components that co-operate in delivering an image query to the base station by executing specific individual tasks. Each component is discussed in more detail in the sections below.

Camera Activity

Prior to capturing a photo, the mobile phone's screen will display to the user what a camera application usually does, the data collected by it's visual sensors. Therefore, since this class requires to constantly draw onto the screen it was designed to extend the Activity component. For clarity and astetical reasons, the activity is displayed on a full screen removing an side bars and hiding all the buttons and layouts. Underneath the display, the class is required to gain access to the built-in camera, therefore, relevant permissions to the camera must be added to the application's manifest in order to allow unrestricted access to the camera. The android system uses information declared in AndroidManifest.xml file in order to allow the application to execute. These information declare permissions the application must have in order to access protected sections of the API [25]. Failing to define certain permissions will cause the application to crash.

Once the photo is captured, data is stored onto a bitmap. This bitmap is used by the application for all proceeding tasks. It is either compressed into a .PNG file prior to cropping or into a .JPG file for local storage. The bitmap is as well used to draw onto the activity's canvas in order to display the image to the user. Suitable instructions usually follow the display of the image in order to guide the user through the next steps.

Networking Service

As previously discussed, all components that require to draw onto the display must extend an activity regardless of how much of the display will be used. In other words, anything that does not wish to draw onto the display doesn't have to be an activity, it can be a service. Since the networking class aims to manage all incoming and outgoing data and does not contribute to the application's GUI, it will extend the service component. Implementing the networking class as a service allows other applications to avail of the

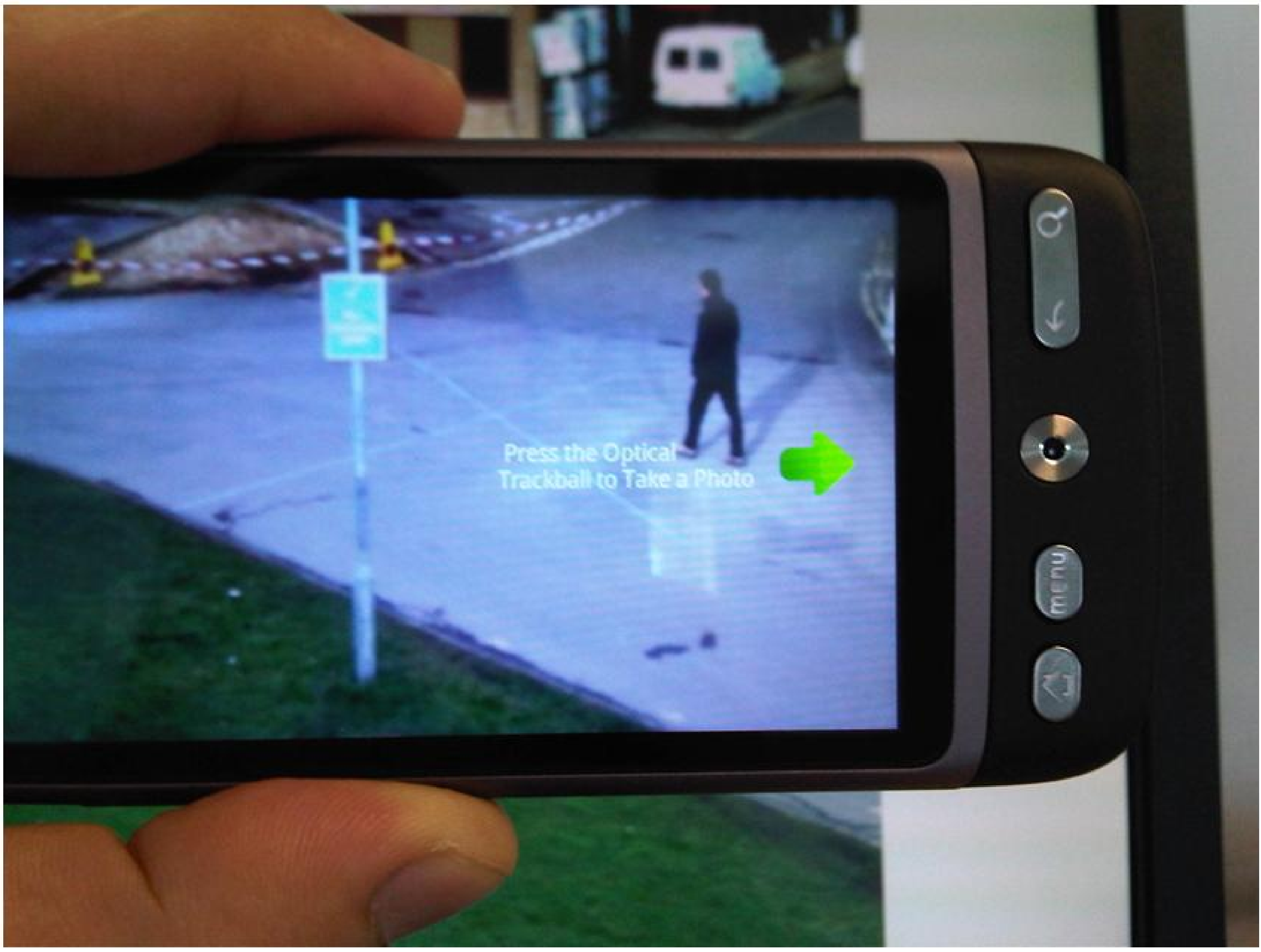


Figure 3.4: Mobile Application GUI.

features provided by this class by simply binding to it. The networking class essentially creates a datagram socket through which datagrams are transmitted to/from the server. Two methods that overload one another are defined within this class. These methods take different types of arguments, either an array of bytes or a string. It then encapsulates the data along with a UDP header and transmits it to the base station. The methods incorporate a size checking mechanism which segments the data into smaller fragments in order to meet the requirements of the UDP protocol in relation to the size of the data to be added to the datagram. Suffixes and prefixes have been added to the data prior to transmission in order to ensure ordered delivery of data and to prevent the loss of datagrams as well as the arrival of duplicates.

Image Editing Tools

The image editing tools designed for this application allows the user to crop the photo containing the target. Cropping assists in creating image queries that focus more on the target and removes most of the background the can be considered as noise. Cropping the image can is easily done by bounding the target inside a rectangle. Using his fingers, the user will mark the coordinates of the bounding rectangle. Visual feedback guides the user through-out the whole process. the application also handles user error, allowing him/her to discard or alter their selection before cropping and sending the image.

Layouts using XML

XML is usually used as a format for data on the Internet. Retrieving and sending data to/from the Internet will normally involve dealing with XML data. In addition, XML provides developers with an intuitive tool for building astetically pleasing graphical user interfaces. The GUI and animations that meet the user when using this application was mostly implemented in XML. The toasts, buttons, layouts and animations have all been defined, aligned and organized in XML files. Figure 4.4 in Section 4.1.1 gives an example of an XML file used to implement layouts different android components. The layout is

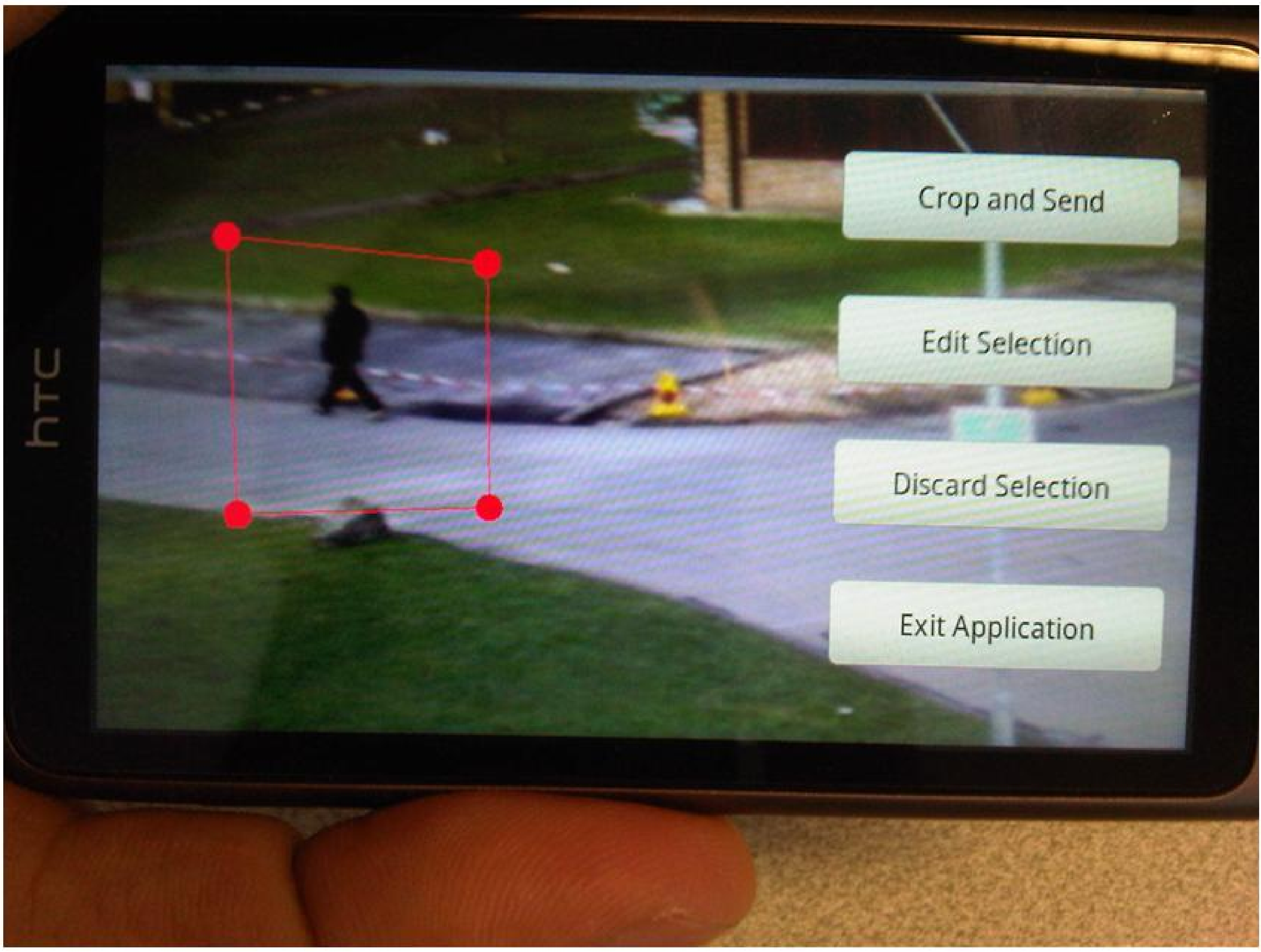


Figure 3.5: Mobile Application Image Editing Tool.

then referenced from the main activity in Java allowing it to easily integrate with the rest of the code.

3.3 The Base Station

The base station is a multithreaded server that accepts both UDP and TCP requests from the smart phones and camera nodes respectively. The base station, as well, provides a user interface to the user. The interface displays updates as they are received from the client camera nodes. Updates include information related to the location of the target, snapshots of the most recent sightings, and the direction the target is moving in. The server is implemented using the Java programming language.

3.3.1 Multithreaded Server

A multithreaded server designed to accept both UDP and TCP requests acts as the system's base station. UDP was the protocol of choice to communicate with smart phones willing to transmit an image query to the base station. The decision was based on the assumption that a mobile phone is very unlikely to transmit more than once to the base station, and the transmission duration will be for very shot due to the size of the data being transmitted. Furthermore details is these regards is given in Section 3.5. As for the TCP protocol, it's used to establish a somehow permanent connection with the client camera nodes. There will be a regular exchange of information between the two nodes, and any transmission is required to be as error free as possible. Therefore, building on top of a robust and connection oriented protocol, such as the TCP protocol, that deals with different types of errors provides a better middleware service.

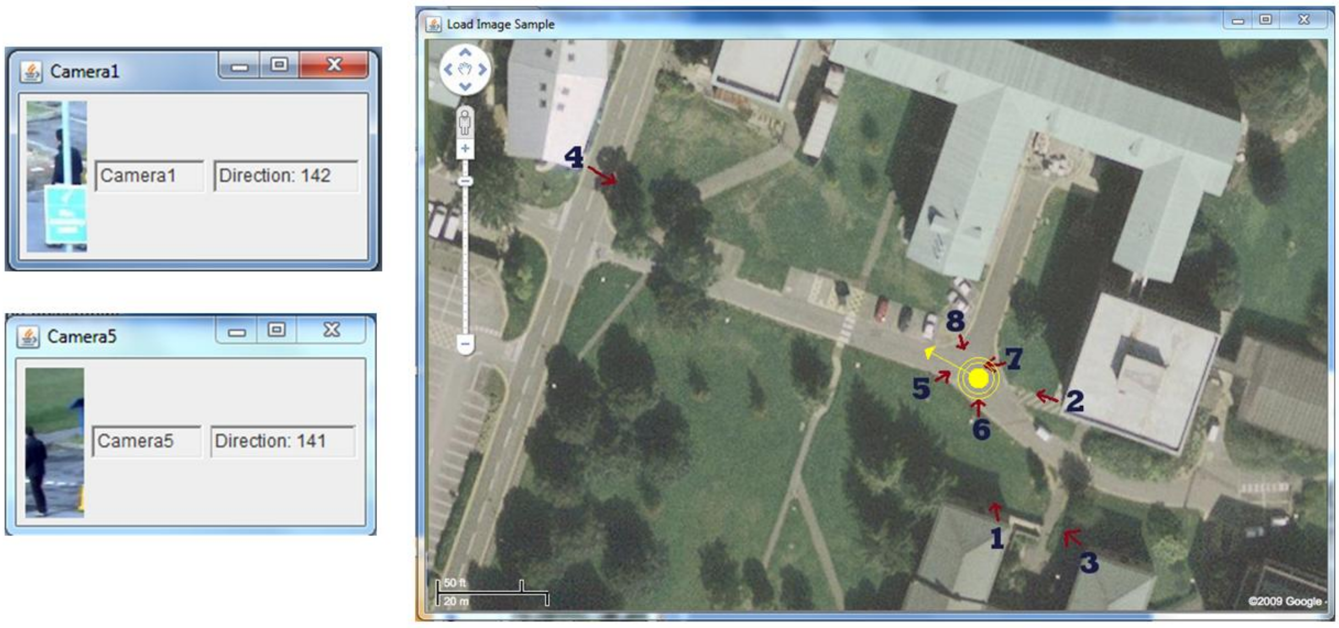


Figure 3.6: Base Station GUI.

3.3.2 Base Station Monitoring User Interface

The base station is not entirely about handling connections from the various system components. There is another aspect to it that aims at providing the user with visual feedback about the system. Users at the base station are presented with a map, displaying the geographical locations of all connected client camera nodes. This is accomplished by mapping camera ID's uniquely assigned by the server with the geographical locations of the client camera nodes. New camera nodes wishing to join the network will be assigned a unique ID and will be requested to supply the base station with its exact geo-locations. Once connected the new camera node will be shown on the map with the rest of the camera nodes. The server remains idle as long as there are no new connections or any image queries received. Once an image query is received, a window pops up on the screen displaying the image. The image will then be distributed to the client nodes to be used as a template image while searching for the specified target. Once the target is detected, the user is informed of the sighting. This will be followed by visual presentations on the map showing the estimated location of the detected target, with respect to the observing camera. Continuous updates are received through-out the tracking process. The value of the angle is displayed which estimates the direction the target is moving in. The tracking process will be accompanied by snapshots of the target as he appears in the camera's field of view to allow the user to rule out any false positives of candidates who've been falsely identified as the target. The target may be seen by more than one camera at once. Therefore, the server may be receiving data from different cameras. All the incoming data from different camera nodes is correlated at the server in order to present valid and consistent information to the user prior to being displayed.

3.4 Client Camera Node

The camera client node is composed of a smart camera that is equipped with high processing power and networking capabilities. It could either be one of the new high tech surveillance cameras that has everything in one apparatus. Or, in our case, could be a desktop computer with a web cam connected to it. In the case of the latter, two application will run on the client camera node. One is Java based application that is used to connect this camera to a network of other cameras and to the base station. It will establish a TCP connection between the camera node and the base station. While the second application will provide all the vision functions that will allow the processing of the collected footage for detection and tracking purposes. It is implemented in C using the OpenCV library.

3.4.1 OpenCV Library for C

The platform used for developing the computer vision section in this project is the OpenCV libraries under Microsoft Visual Studio. OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed for real time computer vision [27] . It has been initially developed by a team from Intel and is currently supported by the Willow Garage. The library has more than 2500 optimized algorithms. Its uses range from interactive art, to mine inspection as well as stitching maps on the web through advanced robotics.

3.4.2 Detection and Tracking - Vision Component

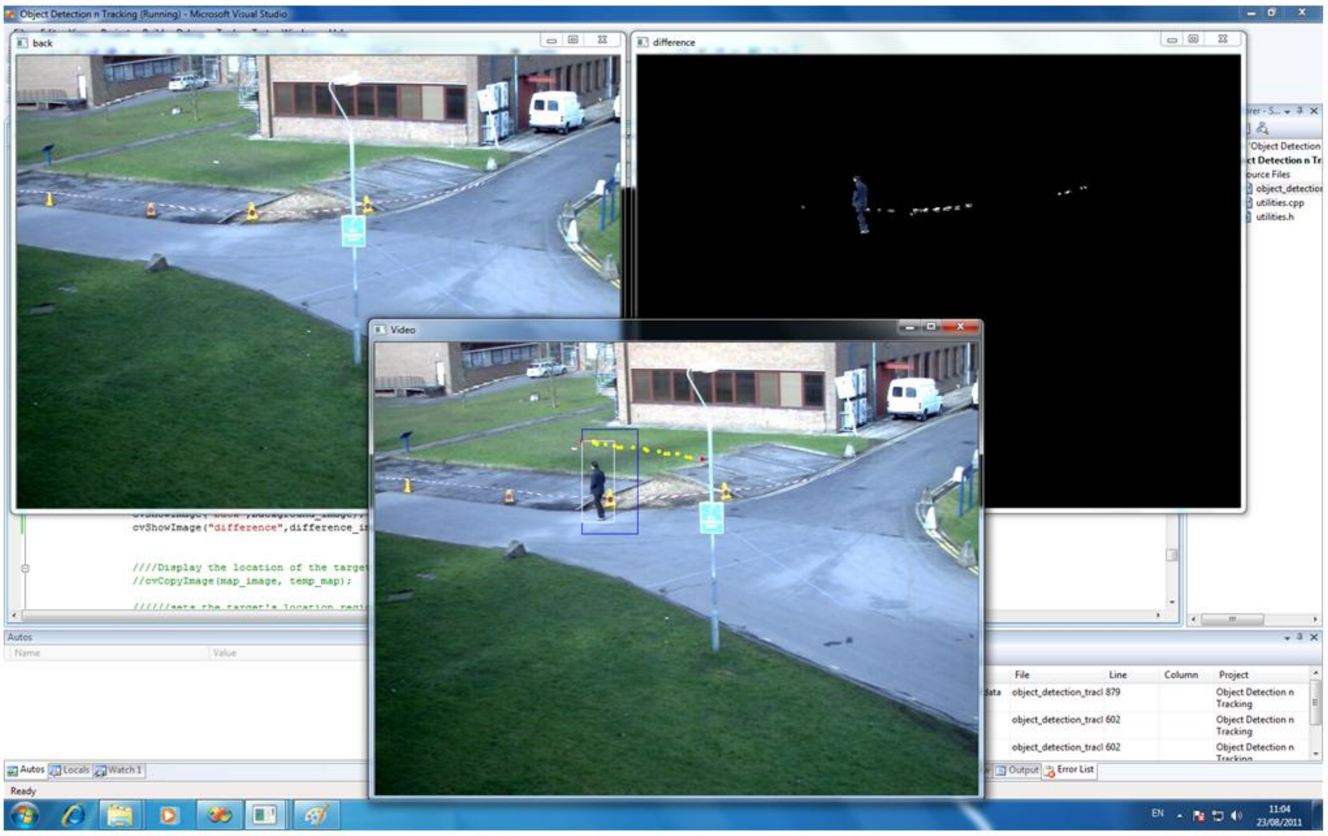
The computer vision component of the client camera node is responsible for the detection and tracking aspect of the whole system. A combination of computer vision techniques that execute in a particular sequence make such a task possible. Computer vision is considered to be quite complex due to the loss of information frames suffer from during recording a video. This happens since a frame is in 2D to represent a 3D object, how images are interpreted, different types of noise, large sizes of data produced while recording a video, level of brightness and many more. This section focuses on the adopted vision techniques, the reason behind each decision and how they contribute in the detection and tracking process.

Introduction to Computer Vision

Computer vision allows machines to interpret images in a similar manner to the way human beings do. This is done by using a combination of vision techniques that execute in a particular sequence. Each of these adopted techniques are used to accomplish a specific task. Before getting into the details of each technique and what its used for, it's preferable to introduce a few popular vision terms that will be used regularly through-out this section. Since this system only deals with videos, a brief introduction on how videos are processed in computer vision will follow. When recording a video, the recording device does that by capturing frames of the field of view of the camera at a certain rate, this rate is normally referred to as fps (frames per second). Therefore, when processing a video each frame is processed individually.

Background and Difference Image Calculation

Looking at each frame individually, it is almost impossible for a machine to be able to identify and label objects in a scene, or even know which objects are moving and will no longer be in the scene any more. Therefore, it is required to have a background. A background is an image of a specific scene that shows all stationary objects in that scene without any moving objects that exist temporarily within the scene. Therefore, a background image of a street somewhere in a city, will only contain the buildings on that street, the street, poles and other objects that are always there, excluding any people and cars. In most of the cases, a background image may be given or easy to acquire. However, this is not always the case. Therefore, whenever a background image cannot be acquired directly from a frame, it has to be calculated. Even if a background is given, it doesn't mean that the background will always be the same forever. Places change, the weather changes and the time of the day changes, thus affecting the captured frames in many different ways. Therefore, calculating backgrounds must always be performed.



```
cvShowImage("difference",difference_i

////Display the location of the target
//cvCopyImage(map_image, temp_map);
```

	File	Line	Column	Project
data	object_detection_tracl	879		Object Detection n Tracking
	object_detection_tracl	602		Object Detection n Tracking
	object_detection_tracl	602		Object Detection n Tracking

Figure 3.7: Difference and Background Image Calculation at Client Camera Node.

Calculating backgrounds can be done in several ways, such as Static Backgrounds, Mode Background image, Median Background image, and the method of Stable Values. The method of static background is when a frame of scene is captured with no moving objects in it. This works, however it doesn't accommodate well to changes within the scene as previously discussed. Next is the Mode Background method, this method requires the machine to keep track of a certain number of frames, and then pick the most common pixel values for each point in these frames. It then puts theses values into one frame. The problems from calculating a background using such a method can be easily identified. First, it requires a lot of processing and a lot of storage since it must keep track of a certain number of frames. Second, the mode in a sequence of frames does not rule out the fact that the object may be temporarily stationary. Another method is the Median Background method, this is very similar to the method previously discussed, but instead of calculating the mode in the previous frames it calculates the medians. This method suffers from similar issues as it's predecessor, as well as when calculating the median, a totally new value is calculated for a pixel that may have not existed in the first place. Therefore, the background calculated will be an image of a scene different than the one calculated for, more likely not real. Last but not least, is the method of Stable Values, the method adopted for calculating the background in this system. The algorithm for stable values is as follows:

- Initially start with the first frame as the background
- Update points in the background image if the value is different in the current frame and has been near this different value for a certain number of frames

Now that it's clear how a background is calculated, the next step will be to explain what the background image is used for. The background image is used to identify moving objects in a scene. This is done by calculating a difference image. A difference image contains only colored pixels that have different values than these in the background image, while the rest of the pixels have a zero value (black). Calculating the difference image is kind of trivial. First you subtract the current frame pixel value for each point in that frame from the corresponding point in the background image. The value is then compared against a certain threshold, to determine which pixels represent the moving object and to, as well, filter out any noise. Most of the algorithms to follow will be applied on the difference image, to improve the speed at which the program executes since any black patches are very likely to be left out.

Object Detection using HBP

Object detection is a very common task in computer vision. It is mainly used to detect objects within a frame whether moving or stationary of different shapes and sizes. There are several methods in which this task may be implemented in, the method normally depends on the type and quality of the input frame, the type and quality of the given template and the target's characteristics. Some of the most popular object detection algorithms include template matching, chamfer matching, statistical pattern recognition and HBP (Histogram Back-Projection). Only template matching and HBP will be discussed in more details as the rest of the algorithms require a much lower level detail (higher resolution images) to give accurate results which is not the case for this system, as CCTV footage contains a large amount of noise.

As for template matching, it may be defined as a technique in which a sub image is searched for within an image through direct comparison to a given template [21]. The region of interest, the sub image, represents the target object, defined by the given template, in 2D. Such a technique is known for it's simplicity, and may be very effective in many cases, especially when the target in the template closely resembles how the target will look like in the searched image. However, if this wasn't the case, template matching is very likely to perform poorly and generate a large amount of false positives. Some of these cases include the following:

- image quality is poor

- different brightness levels
- object of interest is of different size due to the position of the target with respect to the camera
- object of interest is captured from a different view in the template than in the image
- object of interest is at a different angle

and many more. The issues mentioned above are some of the most relevant to issues this system may suffer from if it were to use template matching for object detection. However, many of these issues can be solved, by applying geometric transformations to the template prior to comparison, such as rotation, skewing, change of scale (expand/shrink), panoramic distortion, perspective projection and many more. Comparing a template to an image at every point in that image is computationally expensive on its own. What if you were to apply the mentioned transformational algorithms, which have hundreds of variations each, prior to executing the template matching algorithm. The answer would be that such an algorithm is infeasible, especially with the low detection rate and the high computational costs. Besides that, such a system may never be used for any real time applications.

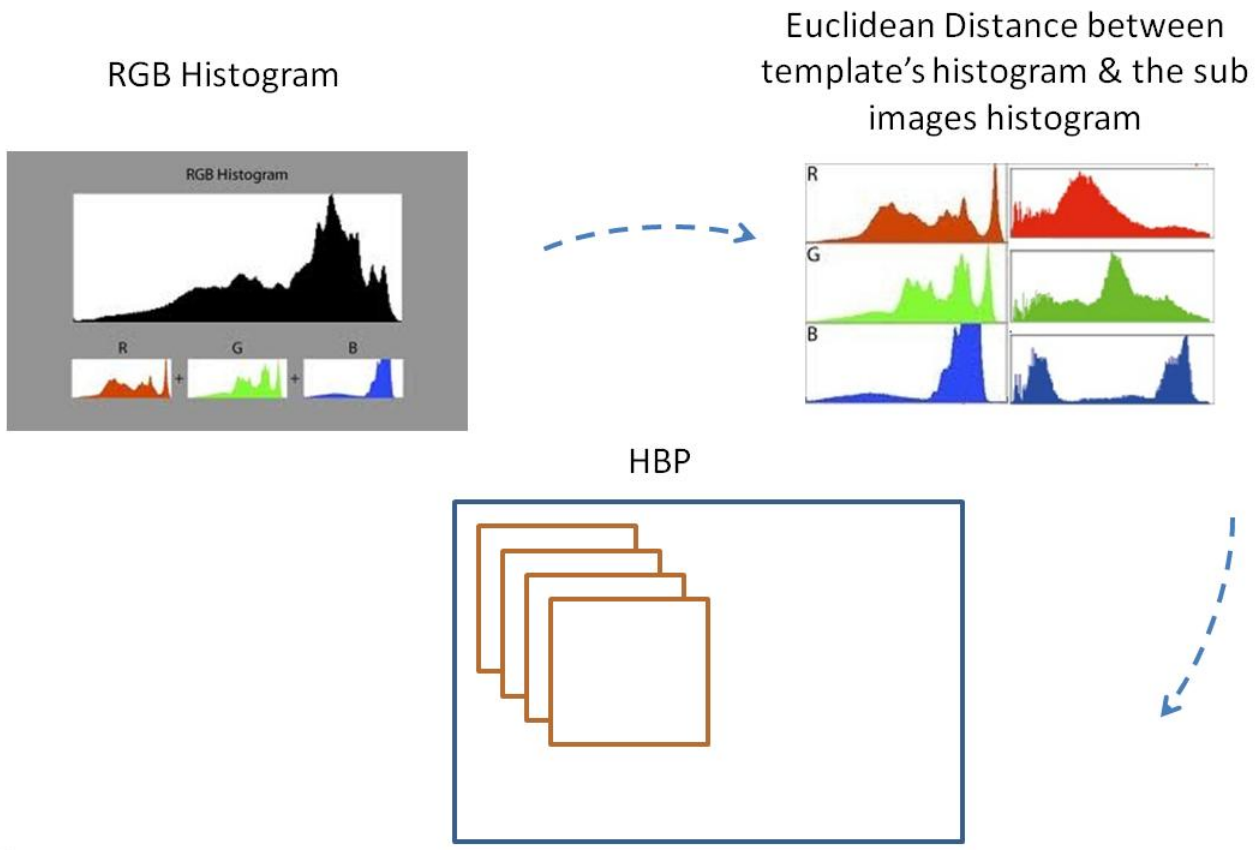

RGB Histogram
RGB Histogram
R
G
B
Euclidean Distance between template's histogram & the sub images histogram
R
G
B
HBP

Figure 3.8: Object Detection using histograms and HBP.

The solution may lie in the other algorithm, HBP. The HBP algorithm slides a window through the current frame and calculates the likelihood that each point belongs to the model defined in the template [20]. The comparison metric it computes the likelihood on is histograms. HBP is applied in different ways, however, you are always required to compute the histograms of both images. Before discussing the different HBP techniques available, it is essential to mention that using histograms to detect an object is a very sensible decision for a system like this. First, image queries created using mobile phones are very unlikely to show the suspects features. Second, the angle the photo is captured at is very likely to be different to that the camera might see the suspect at. Third, detecting the target with regards to the colour of clothing he/she is wearing is very reasonable, even a human would do it such a way if he/she knows nothing about a target they are searching for. Therefore, using the pixel colour values is very likely to produce valid results. Some of the widely known HBP techniques are the following:

- Confidence Back-Projection: Assigns a likelihood value for each point in the image by computing the confidence histogram h$_r$ which is the ratio of the model histogram h$_q$ and the target histogram h$_t$.

$$h_r[m] = \frac{h_q[m]}{h_t[m]}$$

(3.1)

- Quadratic Confidence Back-Projection: is similar to Confidence Back-Projection with the addition of A$_{m,n}$ that defines the similarity of the histogram elements at m & n.

$$h_r[m] = \frac{\sum_n h_q[m] A_{m,n}}{h_t[m]}$$

(3.2)

Smoothing and thresholding are then applied to reduce noise and to reveal the most likely positions of h$_q$.

- Local Histogramming: In local Histogramming the back-projection and smoothing steps are combined. The algorithm computes a local histogram h$_l$ for each point in the target image I[x,y] and measures it's distance to the model histogram h$_q$. After thresholding, the shortest distance indicates the closest match.

$$I''[x,y] = d_{q,l}\left(h_q, h_l(x,y)\right)$$

(3.3)

- Binary Set Back-Projection: points in the target image are assigned a value of true or false depending on the membership in the model binary set s$_q$.

$$I'[x,y] = s_q[k]$$

(3.4)

Smoothing and thresholding are then applied to reduce noise and to reveal the most likely locations of s$_q$.

- Single Element Quadratic Back-Projection: This algorithm combines both Quadratic Confidence and Binary Set Back-Projection, weighing points in the target image based on their similarity to the back projection element m, where s$_q$[m] $= 1$ and s$_q$[k] $= 0$.

$$I'[x,y] = A_{m,k}$$

(3.5)

Smoothing and thresholding are then applied to reduce noise and to reveal the most likely locations of s_q.

The algorithms defined above were tested and evaluated by John R. Smith [20] to find the most effective of them all. It was found that Quadratic and regular Confidence Back-Projection's main drawback is that the back-projection algorithms can only be applied to like bins. Local Histogramming proved to be very robust. Binary Set Back-Projection was very simple but results were not as adequate as Local Histogramming. The same applies to Single Element Quadratic Back-Projection. Based on these tests, Local Histogramming was chosen as the type of HBP to be used for this system.

Frame Segmentation

Applying HBP using the Local Histogramming method proved, to be very computationally expensive. If you were to break down the steps involved in applying this algorithm, you would notice that at every point in the target image you are to calculate a histogram of a sub-image that is of the same size as the template image, then measure the distance between the sub-image's histogram and that of the template's image. Doing this for large frames in the videos that are recorded at around 15 fps (low resolution), will take extremely long and is totally infeasible. Therefore, in order to solve this problem, each frame was segmented into smaller overlapping sub-images of different sizes, yet larger than that of the template image. In this case it was around 61 sub images. This means, if the frame size was around 700x700 and the template image size was around 200x200, instead of computing histograms for 501x501 points in the image and then measuring their distance to the model histogram, this will only be done 61 times for each frame, i.e. around 4000 times faster.

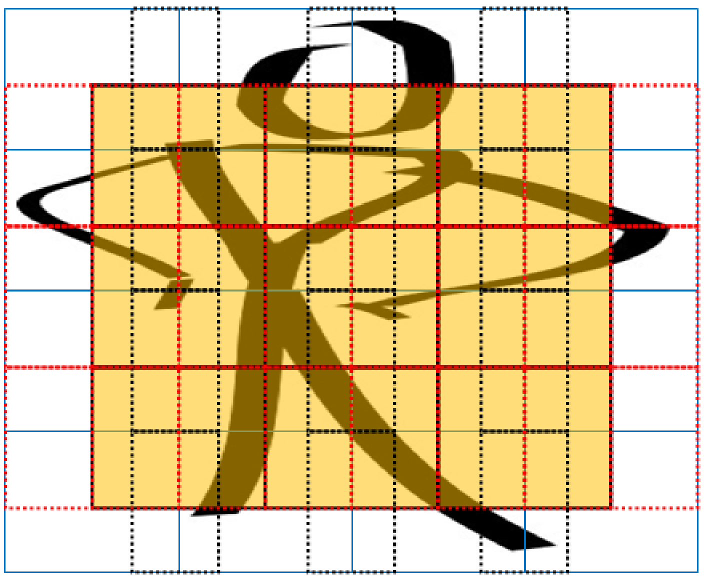


Figure 3.9: Image Segmentation Diagram.

Euclidean Distance

In order to measure the distance between two histograms, the method of Euclidean distance was used. The Euclidean distance between two points can be defined as the shortest distance between these two points. The Euclidean distance equation used to measure the distance between two histograms is as follows:

$$d(H_1, H_2) = \sqrt{\sum (H_1(x,y,z) - H_2(x,y,z))^2}$$

(3.6)

Object Tracking Using a Kalman Filter

Once the target is detected, it is no longer required to be detected again and again for every new frame. A better practice would be to track the detected target in

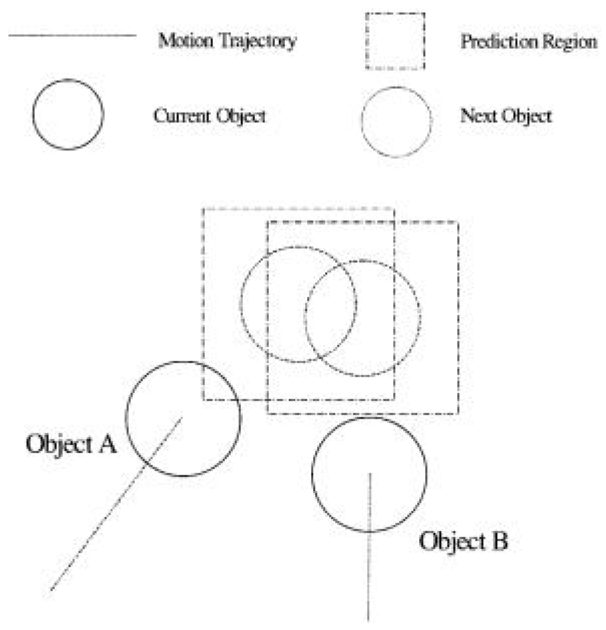


Figure 3.10: Kalman filter when applied to predict the next possible position of an object based on it's current position.

every new frame based on the previous location he/she were detected in. Therefore, once the target is detected in the current frame, the position it is likely to be at in the new frame can be efficiently predicted using Kalman filtering [28, 11]. The Kalman filtering algorithm estimates the state of the system by taking into consideration a set of measurement errors. It assumes that the system is linear, and the observations of it are linear functions of the underlying state[21]. Considering two sequential observations (z_1 with a variance σ_{z1} and z_2 with a variance σ_{z2}), basic Kalman mathematics can be explained as follows: The system state $x(t_2)$ is:

$$x(t_2) = z_1 + \left[\frac{\sigma_{z1}^2}{\sigma_{z1}^2 + \sigma_{z2}^2}\right][z_2 - z_1]$$

(3.7)

Substitute the observation with the previous model and bringing the Kalman filter form into the equation K(t), where

$$K(t) = z_1 + \left[\frac{\sigma^2_{z(t-1)}}{\sigma^2_{z(t-1)} + \sigma^2_{zt}} \right]$$

(3.8)

gives the following

$$x(t_2) = x(t_1) + K(t_2)[z_2 - x(t_1)]$$

(3.9)

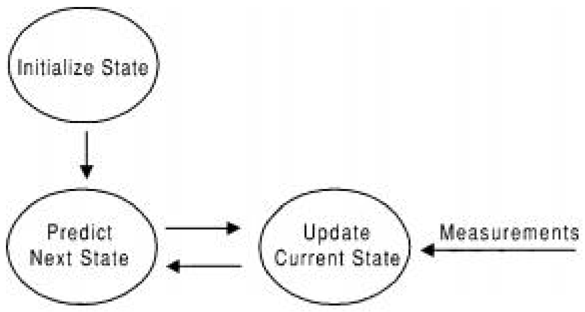


Figure 3.11: Diagram showing the method in which a Kalman filter predicts a system's next state from the current state.

For this system, Kalman filtering was used to estimate the next likely positions the target might be at. Since it is assumed that people move at a certain speed, a rectangle of a certain size that is a few human steps larger than the size of the detected target will be used to limit the search for the target in the new frame. Therefore, Kalman filtering allows the system to narrow down the region in which HBP will be applied producing more accurate results and less computational costs.

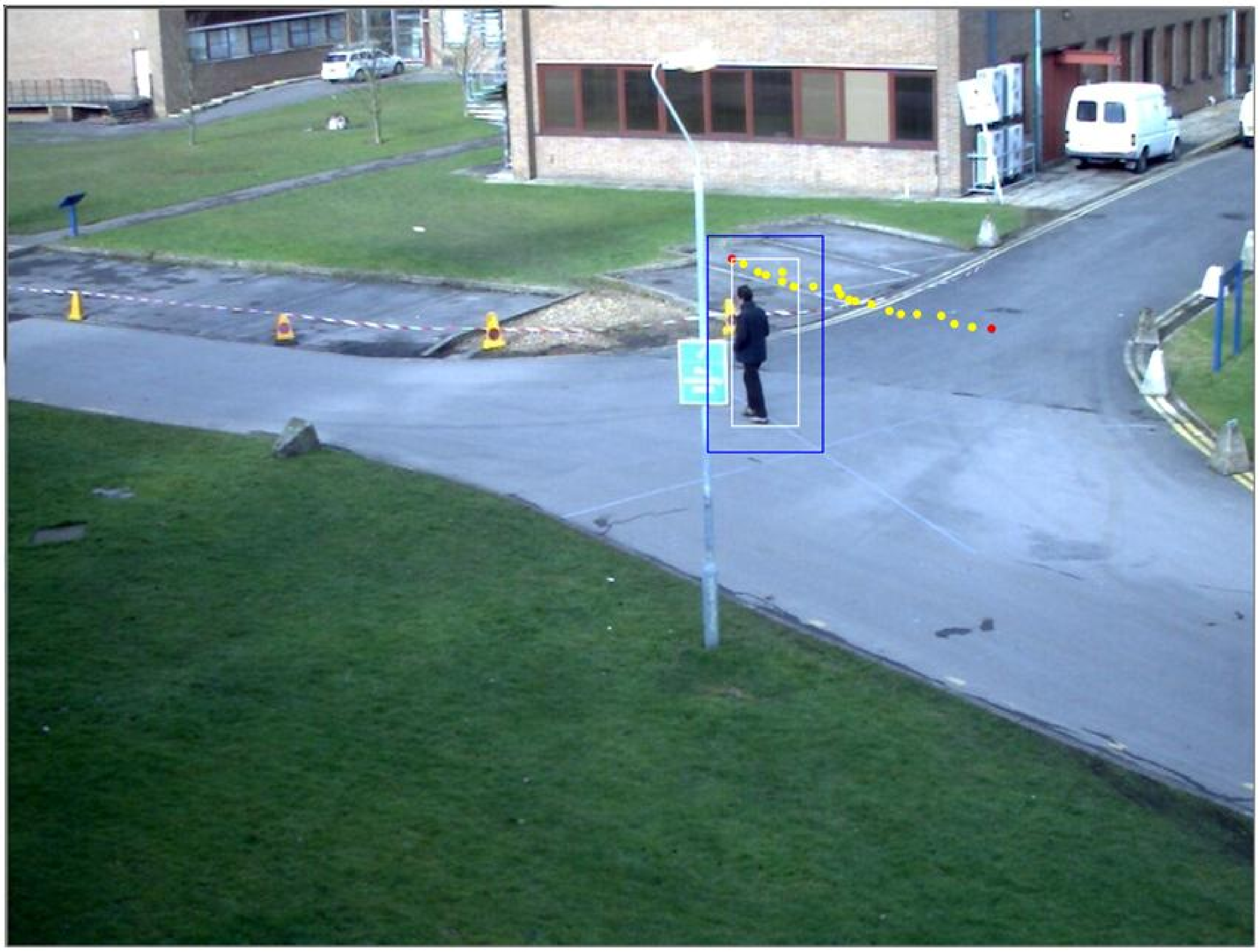


Figure 3.12: Displays how the system uses a Kalman filter to track a target from one frame to another.

Size Thresholds

Size thresholds have been applied as well to the detected objects. This produces more accurate results by ruling out objects that have similar histograms to the object model but are either too small or too large to be the object. Taking into consideration, the position of the camera and how large and small human beings may appear when viewed from this camera, such a threshold helps removes moving objects smaller or larger than human beings that have similar histogram ratios, such as cars or animals.

Direction of Movement Calculator

In order to evaluate the direction the target is moving in, it is required to calculate the angle of the direction vector that indicates the direction the target is moving in. The direction vector can be acquired by keeping track of the targets previous locations as well as the current location. Taking older previous locations rather than the most recent one will provide more accurate results.

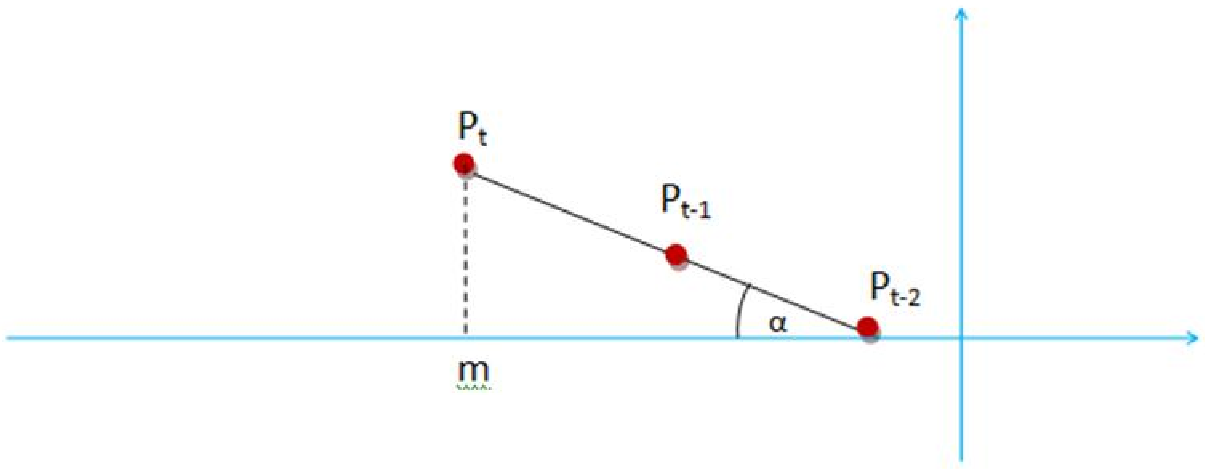


Figure 3.13: Method of Direction Calculation.

$$\sin(\alpha) = \frac{\sqrt{(y_m - y_{p(t)})^2}}{\sqrt{(x_{p(t)} - x_{p(t-2)})^2 + (y_{p(t)} - y_{p(t-2)})^2}}$$

(3.10)

3.4.3 Middleware Component

In order to allow the camera nodes to communicate with the base station, a middleware component is required. The middleware component will be designed on top of the TCP protocol. It will allow the camera nodes to identify what kind of data it is receiving from the base station, reassemble it if it was segmented prior to sending and prefix data prior to sending. The middleware component is a key aspect of the collaboration of the system, otherwise each camera will be considered autonomous from the whole system.

3.5 System Middleware

This section discusses how the middleware components of the individual elements of the system are interconnected.

3.5.1 Network Topology

This section discusses the network topology chosen for the layout of this system. A network topology is the pattern followed for the layout of different entities in the system or a network. It describes the structure of the network and how these components are linked together. For this system, it was found that a hybrid topology would be the more efficient especially due to the fact that the system must be expandable and scalable. The hybrid topology adopted combines both a star topology and a distributed one, where each of the end nodes in a conventional star topology is a central node for another star topology or cluster that branches at it, the diagram below provides a better overview of what the topology looks like. However, for this system, the latter topology was not adopted due to the fact that the prototype will not include as many nodes as there would have been in the real system, thus affecting performance. Therefore, a conventional start topology was followed for this purpose, thus producing better results for a system of the described scale. The 2nd figure is a visual representation of the adopted topology.

3.5.2 Communication Protocols

This chapter covers the architecture of the system from the networking layer that underpins the whole system, to the different applications that make use of this network layer to communicate with one another such as the camera client nodes and the mobile application.

For this system, a combination of Internet protocols were used to satisfy the heterogeneous components forming this system as well as the various functionalities they are expected to provide. The Internet protocols mentioned are mostly those that lie within the transport layer of the OSI model (Open System Interconnection model). The diagram below shows the OSI model emphasizing the location of the transport layer within this model. This is then followed by a brief description of the two main protocols in this layer and how these protocols have been adopted by this system.

TCP/IP

TCP (Transmission Control Protocol) is a connection oriented protocol that was particularly designed for the purpose of providing a reliable end-to-end data stream over an unreliable network[23]. Normally, when transmitting data packets between two nodes, the packets propagate through the Internet-work. The Internet-work is typically formed of several networks connected together, each with different topologies, bandwidths, delays, packet sizes and other parameters. The intention of TCP was to provide a service that is capable of dynamically adapting to the characteristics of the Internet-work and that is robust in the face of failures. The TCP protocol is part of the Internet Suite Protocol in the transport layer. Establishing and Terminating a TCP connection between two nodes is normally accomplished using the three-way and four-way handshake phases respectively. During transmission, each sent packet is is created by encapsulating a TCP header with data to be sent. The TCP header contains valuable information that is required for the complete and successful delivery of a file from one end to another. The protocol also provides features such as error detection, flow control and congestion control, making it a reliable service.

UDP

UDP (User Datagram Packet) is a connectionless transport protocol that is supported by Internet Protocol Suite. The Internet Protocol Suite is the set of protocols used for the Internet[23]. UDP allows applications to transfer encapsulated data packets, usually known as datagrams, without the need to set up a connection. It provides an unreliable service, therefore, the protocol does not handle issues such as datagrams arriving out of order, being lost or even receiving duplicate packets. Such a protocol is considered useful when used with servers that are required to handle small queries from a huge number of clients.

Uses

For this particular system, there are two networks or in other words two types of connections. There is one that connects the mobile application to the server and another that connects the server to the client camera nodes. Each of these connections has its own characteristics and will be used for different purposes, therefore, it would be a sensible decision to assign each with a different protocol. Taking in to consideration, the merits and demerits of the previously described protocols, and due to the reasons to follow, the UDP protocol was picked to be used for transmitting the image query from the mobile phone to the server while the TCP protocol will be used to connect the camera nodes to the server. For clarity purposes, it would make more sense to point out the fact that the server is a multithreaded server that has been designed to accept both TCP and UDP requests.

In any conventional scenario, a mobile application will only be transmitting one image to the server. Taking into consideration that additional measures are taken to ensure the guaranteed delivery of the transmitted packets to the server and the ordered delivery of these packets, as well as the fact that this mobile application is unlikely to be transmitting again to the server, UDP is the protocol of choice. Therefore, establishing a connection between the mobile phone and the server through a three way handshake is considered inefficient. It may as well effect the use of such a system for real time purposes due to the delay it introduces for establishing and tearing down the connection. This will have a further effect on getting the image to the client camera nodes, thus postponing the launch of the search for the target at the camera nodes.

As for connecting the client camera nodes to the server, a protocol that is more reliable such as TCP/IP has been chosen. The main motivation behind such a choice is supported by the fact that there will be regular data transmissions between the connected nodes. Therefore, it is required to have a permanent connection between the server and the client camera nodes such as that provided by the TCP/IP protocol. Furthermore, the server holds a map that indicates the geographic location of each of the client camera nodes in their deployment environment with respect to one another and with respect to the server. Such a map in created when a connection is established between the two nodes (the server and any of the client camera nodes). Once a connection is established, the server assigns each client camera a specific id to which it maps to a specified geographical location indicated by the administrator. This allows the server to multicast an image query to a selected cluster of client cameras depending on the geo-information received with the image from the smart phone. Such information narrows down the regions of interest, or the next possible target locations, thus allowing the detection and tracking process to be executed efficiently, utilizing both the available bandwidth and processing power. It's worth mentioning that implementing such an application on top of UDP protocol is unfeasible and inefficient due to all the associated complexities.

3.6 Software Development Approach

The Iterative and Incremental Model was the adopted software development for building this system. It is considered an updated version of the waterfall model. It makes the early stages of the life cycle, Requirements and Design, less vital than the Waterfall Model, as there is room for change in later iterations. Thus early stages of the software development cycle are not as critical as they are in other development models. This results in reduction in costs induced from inadequate decisions made at early stages. The model divides the cycles according to the required functionalities or with respect to the area of development, if the system is to be developed in different platforms. Using such a model, provides at the end of each cycle a system component that has been designed, built, and tested and is ready for use.It is favored by clients as there's room for modifications, allows regular feedback, aims at providing a prototype at every cycle. This means features not included in the initial set of requirements can be included at later stages. This results in better customer satisfaction on the cost of meeting submission deadlines. Thus the Iterative and Incremental Model provides a dynamic development model that allows the production of robust systems that has been rigorously tested from different aspects and in various methods.

Chapter 4

Implementation

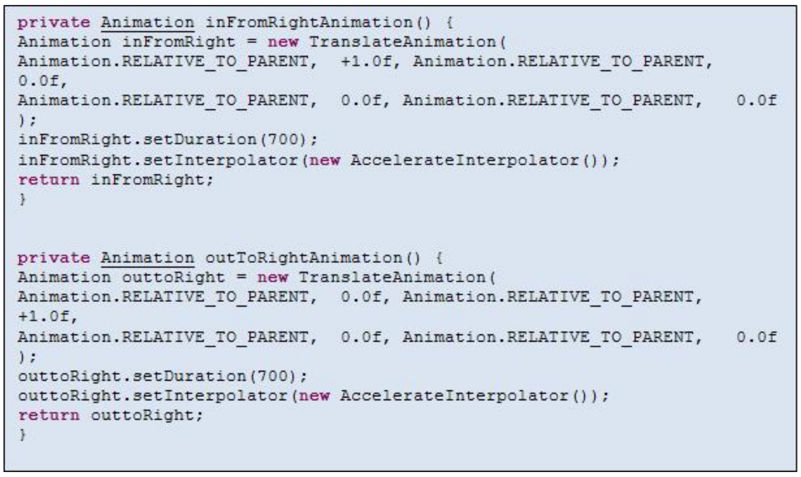
This chapter focuses on how the algorithms discussed in Chapter 3 were applied while building this application. It presents the underlying classes that collaborate in providing a higher level task required by the system. It also presents the UML diagrams that give an overview of how these classes relate to one another.

4.1 Mobile Application

The mobile application was developed using the Android SDK which is based on both Java and XML. Most of the algorithms are coded using Java. XML was mostly used in defining layouts for the GUI, as well as the manifest that allows the configuration of access permissions required by the application.

4.1.1 Main Activity

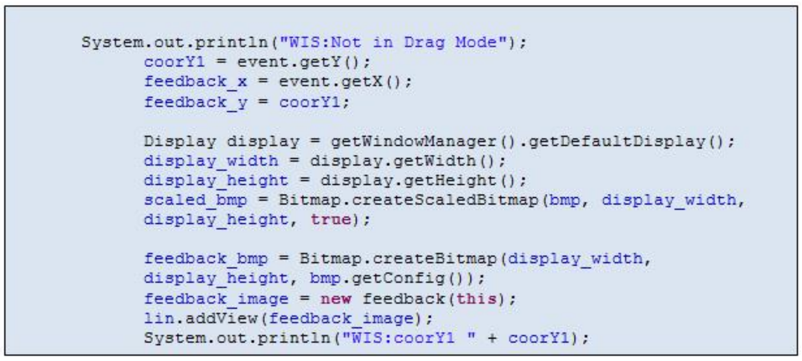
The camera class extends the Android activity class. It's the main activity for this application. It contains code that allows the user to capture and edit a photo. It's the only activity in this application, and makes use of features provided by other classes and services, through either creating local objects or through binding in the case of a service. The figures in this section display of the code used to implement features that will handle touch inputs, crop and display images.



```
private Animation inFromRightAnimation() {
Animation inFromRight = new TranslateAnimation(
Animation.RELATIVE_TO_PARENT,  +1.0f, Animation.RELATIVE_TO_PARENT,
0.0f,
Animation.RELATIVE_TO_PARENT,  0.0f, Animation.RELATIVE_TO_PARENT,   0.0f
);
inFromRight.setDuration(700);
inFromRight.setInterpolator(new AccelerateInterpolator());
return inFromRight;
}


private Animation outToRightAnimation() {
Animation outtoRight = new TranslateAnimation(
Animation.RELATIVE_TO_PARENT,  0.0f, Animation.RELATIVE_TO_PARENT,
+1.0f,
Animation.RELATIVE_TO_PARENT,  0.0f, Animation.RELATIVE_TO_PARENT,   0.0f
);
outtoRight.setDuration(700);
outtoRight.setInterpolator(new AccelerateInterpolator());
return outtoRight;
}
```

Figure 4.1: Android Animation Layout Example.

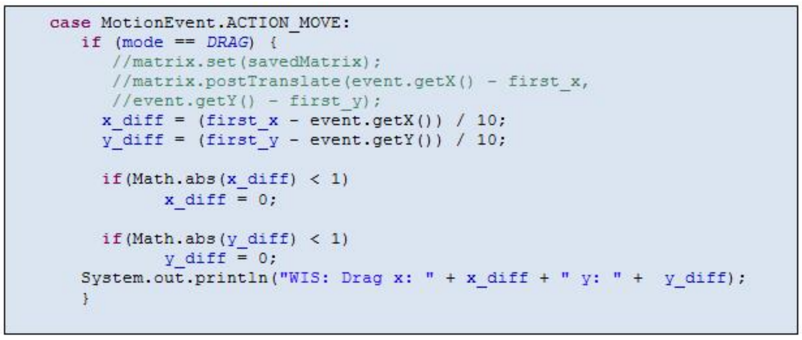
Figure 4.1 defines a few of the animations were used to add visual effects when calling certain layouts. The layout this code was used to be called upon was the buttons layout which allows the user to enter specific modes of the application. These modes will provide environments that support the implemented image editing tools. These animations add a sliding effect to the assigned buttons.



```
System.out.println("WIS:Not in Drag Mode");
      coorY1 = event.getY();
      feedback_x = event.getX();
      feedback_y = coorY1;

      Display display = getWindowManager().getDefaultDisplay();
      display_width = display.getWidth();
      display_height = display.getHeight();
      scaled_bmp = Bitmap.createScaledBitmap(bmp, display_width,
      display_height, true);

      feedback_bmp = Bitmap.createBitmap(display_width,
      display_height, bmp.getConfig());
      feedback_image = new feedback(this);
      lin.addView(feedback_image);
      System.out.println("WIS:coorY1 " + coorY1);
```

Figure 4.2: Android Touch Events Handling Example.

Figure 4.2 defines how the application handles touch inputs from the user. It at the beginning detects whether the user is trying to place points on to the image or edit the location of his/her selection by dragging an already placed bounding rectangle. This mode is usually specified by which buttons has the user selected. If the user was to crop the image through the selection of bounding points, some visual feedback is required. Therefore, the canvas that currently displays that image on the screen needs to be redrawn over recursively. A canvas in android, contains the draw() call that allows a developer to draw in android [26]. The method in Figure 4.2 calls the feedback class that takes the current bitmap as an argument, and draws onto it again, creating a new bitmap containing all the recent inputs and then displays it on the devices screen.

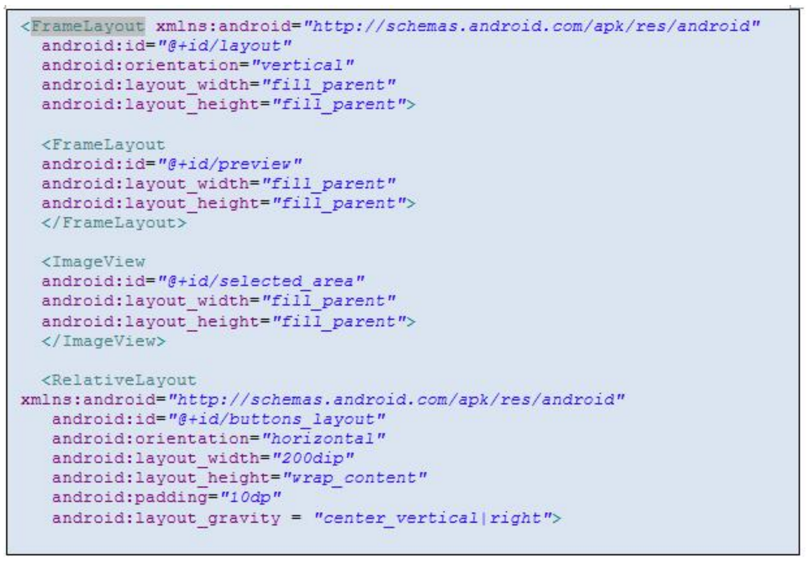


```
case MotionEvent.ACTION_MOVE:
   if (mode == DRAG) {
      //matrix.set(savedMatrix);
      //matrix.postTranslate(event.getX() - first_x,
      //event.getY() - first_y);
     x_diff = (first_x - event.getX()) / 10;
     y_diff = (first_y - event.getY()) / 10;

     if(Math.abs(x_diff) < 1)
          x_diff = 0;

     if(Math.abs(y_diff) < 1)
          y_diff = 0;
   System.out.println("WIS: Drag x: " + x_diff + " y: " +  y_diff);
   }
```

Figure 4.3: Implementing Drag Using Touch Inputs in Android.

Figure 4.3 shows how the drag feature is implemented in this application. This is implemented by receiving regular updates of the current finger positions, every few milliseconds, and calculating the difference between the current coordinates and the previous coordinates, then redraws the rectangle in the new position. Similarly to the previous part, the canvas needs be redrawn with every update, however the same bitmap needs to remain in the background. The layer holding the rectangle needs only to be updated.

Figure 4.4 presents an example of some XML code, it defines how the main camera activity will be laid out. As you can notice, there are many layouts defined in the file. All the layouts are nested inside the main Frame Layout and will be displayed when required by the application. The first layout inside the main layout, is another Frame Layout. However this layout is used to display data received from the camera. This is followed by an image view, in which images are displayed. The third, is a Relative



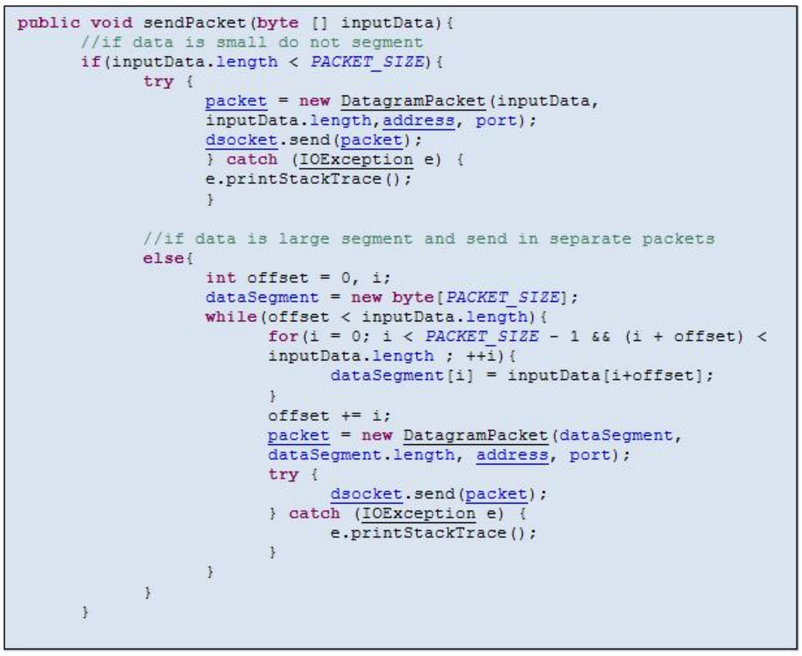
```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/layout"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">

  <FrameLayout
  android:id="@+id/preview"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  </FrameLayout>

  <ImageView
  android:id="@+id/selected_area"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  </ImageView>

  <RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/buttons_layout"
   android:orientation="horizontal"
   android:layout_width="200dip"
   android:layout_height="wrap_content"
   android:padding="10dp"
   android:layout_gravity = "center_vertical|right">
```

Figure 4.4: Android XML Layout Example.

Layout. A Relative Layout gives the developer more control over how components inside this layout are to be aligned than other layouts would. This layout contains all the buttons that will display what options the user have after he/she captures a photo.

4.1.2 Middleware Service

The networking class extends the service class. It allows the camera activity to bind to it once it requires to transmit any data. The class as well contains methods that allow the segmentation of data prior to sending. Figure 4.5 show how the constructor of this class initializes all the variables, defining the base stations address and the port it is to transmit and receive on. It as well shows the method that is used to send a packet. This method overloads another method that takes a string input instead of an array of bytes. The method checks the size of the data, compares it to the maximum datagram size allowed (60 KB) in order to either segment before transmitting or immediately transmit.



```
public void sendPacket(byte [] inputData){
        //if data is small do not segment
        if(inputData.length < PACKET_SIZE){
                try {
                        packet = new DatagramPacket(inputData,
                        inputData.length,address, port);
                        dsocket.send(packet);
                        } catch (IOException e) {
                        e.printStackTrace();
                        }

                //if data is large segment and send in separate packets
                else{
                        int offset = 0, i;
                        dataSegment = new byte[PACKET_SIZE];
                        while(offset < inputData.length){
                                for(i = 0; i < PACKET_SIZE - 1 && (i + offset) <
                                inputData.length ; ++i){
                                        dataSegment[i] = inputData[i+offset];
                                }
                                offset += i;
                                packet = new DatagramPacket(dataSegment,
                                dataSegment.length, address, port);
                                try {
                                        dsocket.send(packet);
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }
                        }
                }
        }
```

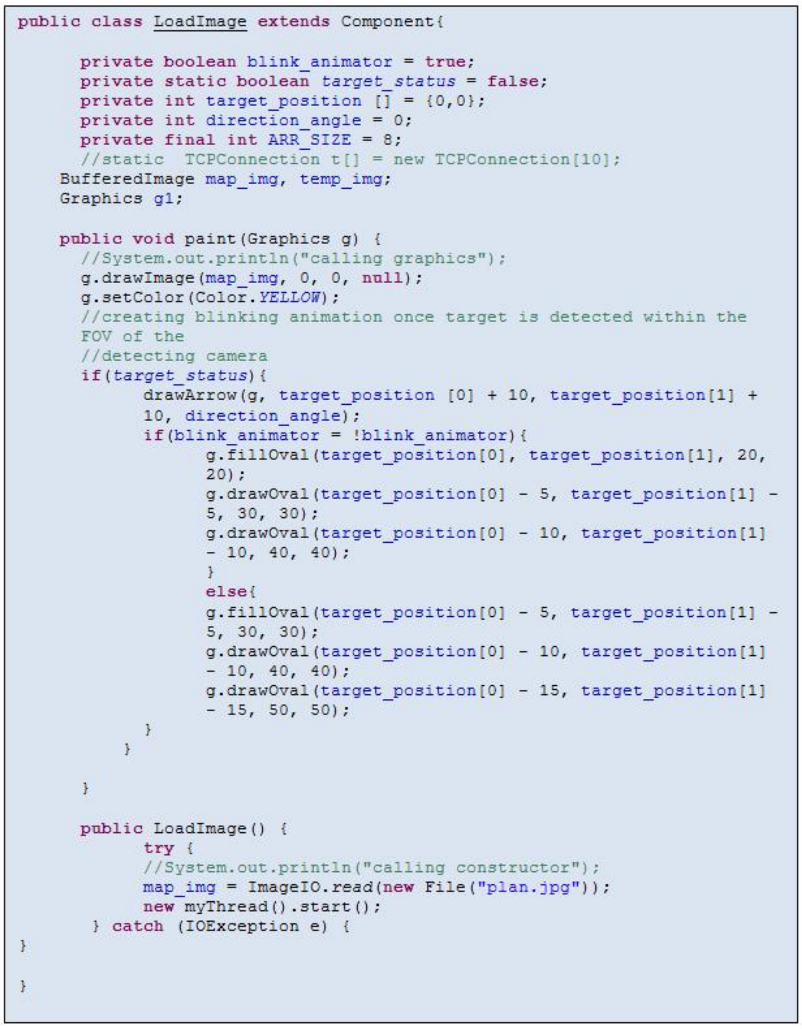
Figure 4.5: Packet Segmentation Method.

4.2 The Base Station

The base station is a multithreaded server that accepts both UDP and TCP requests from the smart phones and camera nodes respectively. The base station, as well, provides a user interface to the user. The interface displays updates as they are received from the client camera nodes. Updates include information related to the location of the target, snapshots of the most recent sightings, and the direction the target is moving in. The section below gives an in depth discussion on how these components were implemented using the Java programming language.

4.2.1 Updates Manager

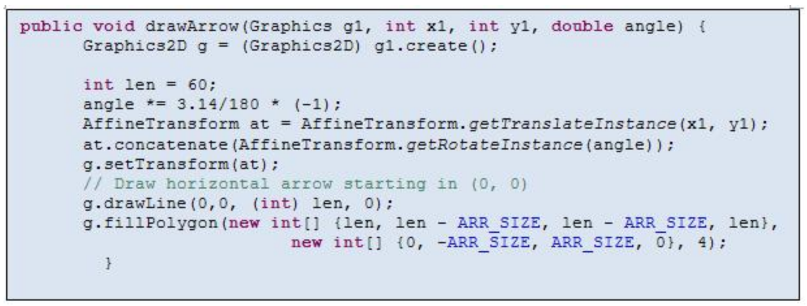
The GUI component describes how the map is loaded into a frame in the main server class, and the image class waits for updates from the camera client nodes and displays the data accordingly. It as well contains a method that draws an arrow indicating



```
public class LoadImage extends Component{

      private boolean blink_animator = true;
      private static boolean target_status = false;
      private int target_position [] = {0,0};
      private int direction_angle = 0;
      private final int ARR_SIZE = 8;
      //static  TCPConnection t[] = new TCPConnection[10];
    BufferedImage map_img, temp_img;
    Graphics g1;

    public void paint(Graphics g) {
      //System.out.println("calling graphics");
      g.drawImage(map_img, 0, 0, null);
      g.setColor(Color.YELLOW);
      //creating blinking animation once target is detected within the
      FOV of the
      //detecting camera
      if(target_status){
            drawArrow(g, target_position [0] + 10, target_position[1] +
            10, direction_angle);
            if(blink_animator = !blink_animator){
                  g.fillOval(target_position[0], target_position[1], 20,
                  20);
                  g.drawOval(target_position[0] - 5, target_position[1] -
                  5, 30, 30);
                  g.drawOval(target_position[0] - 10, target_position[1]
                  - 10, 40, 40);
                  }
                  else{
                  g.fillOval(target_position[0] - 5, target_position[1] -
                  5, 30, 30);
                  g.drawOval(target_position[0] - 10, target_position[1]
                  - 10, 40, 40);
                  g.drawOval(target_position[0] - 15, target_position[1]
                  - 15, 50, 50);
            }
          }

      }

      public LoadImage() {
            try {
            //System.out.println("calling constructor");
            map_img = ImageIO.read(new File("plan.jpg"));
            new myThread().start();
       } catch (IOException e) {
}

}
```

Figure 4.6: Base Station Display Updater Method.

the direction the target is moving in according to the angle received from the camera node and to the camera's ID, which allows the server to identify the location of the camera and the direction it is pointing in. In order keep the map displaying up-to-date information, a separate thread keep repainting the graphics object and the static variables used throughout the tracking process.



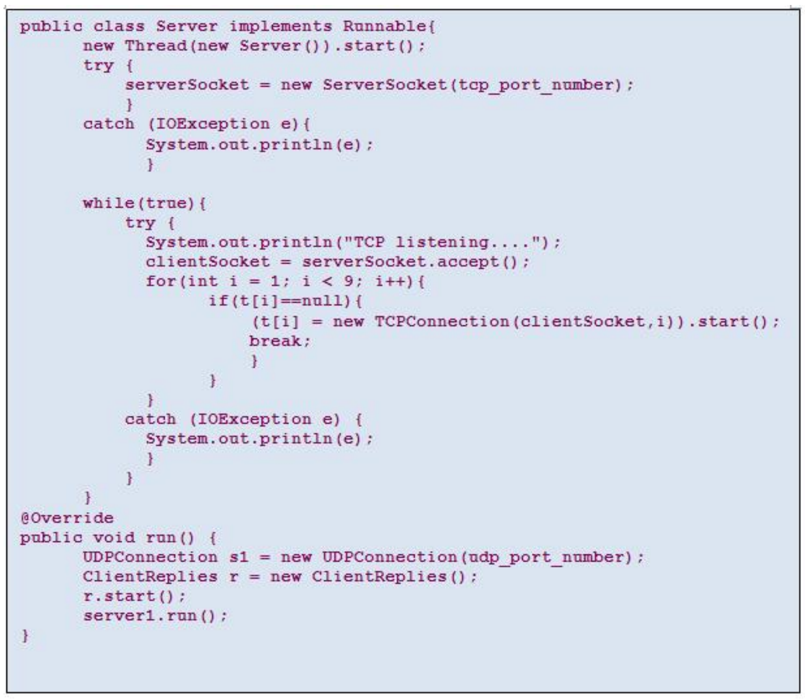
```
public void drawArrow(Graphics g1, int x1, int y1, double angle) {
      Graphics2D g = (Graphics2D) g1.create();

      int len = 60;
      angle *= 3.14/180 * (-1);
      AffineTransform at = AffineTransform.getTranslateInstance(x1, y1);
      at.concatenate(AffineTransform.getRotateInstance(angle));
      g.setTransform(at);
      // Draw horizontal arrow starting in (0, 0)
      g.drawLine(0,0, (int) len, 0);
      g.fillPolygon(new int[] {len, len - ARR_SIZE, len - ARR_SIZE, len},
                         new int[] {0, -ARR_SIZE, ARR_SIZE, 0}, 4);
        }
```

Figure 4.7: Target Direction Arrow Drawing Method.

Figure 4.7 shows how the method that is used to draw the arrow (discussed above) that displays the direction the target is moving in.

4.2.2 Multithreaded Server Application

Figure 4.8 displays the main server class that implements the Runnable interface. Part of the class is shown above displaying the code the handles incoming UDP and TCP requests. As you can see, the TCP server can only handle 8 requests. This is due to the fact that the scenario it was tested in contains 8 cameras, therefore, it was optimized to suit the latter.



```
public class Server implements Runnable{
      new Thread(new Server()).start();
      try {
          serverSocket = new ServerSocket(tcp_port_number);
          }
      catch (IOException e){
            System.out.println(e);
            }

      while(true){
          try {
            System.out.println("TCP listening....");
            clientSocket = serverSocket.accept();
            for(int i = 1; i < 9; i++){
                  if(t[i]==null){
                      (t[i] = new TCPConnection(clientSocket,i)).start();
                      break;
                      }
                  }
            }
          catch (IOException e) {
            System.out.println(e);
            }
          }
      }
@Override
public void run() {
      UDPConnection s1 = new UDPConnection(udp_port_number);
      ClientReplies r = new ClientReplies();
      r.start();
      server1.run();
}
```

Figure 4.8: Multithreaded Server Implementation.

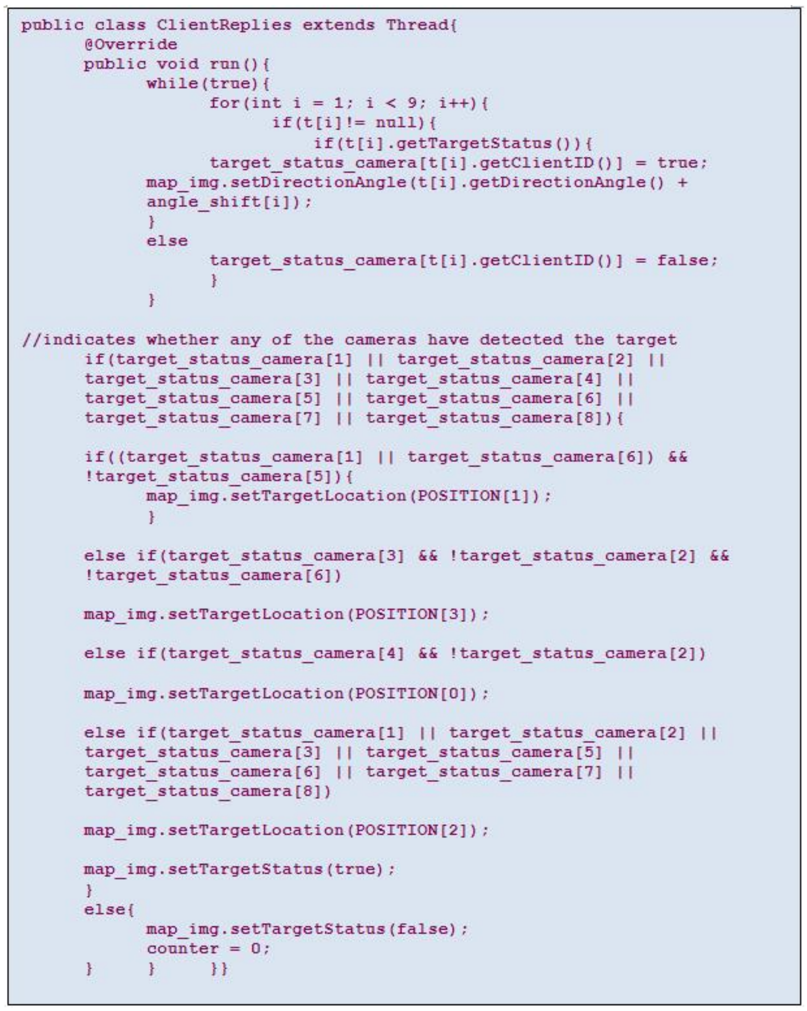
Figure 4.9 shows how the server correlates incoming data from different camera nodes in order to display accurate information to the user. Correlating the data depends on camera related information that has been initialized upon the establishment of a connection with each camera.

4.3 Client Camera Node

The camera client node is separated into two main components, middleware and computer vision. The middleware is implemented in Java while the computer vision is implemented in C using the OpenCV library.

4.3.1 Computer Vision Component

This section illustrates how the algorithms that have been adopted in the design stage, discussed in Chapter 3, were implemented in C using the OpenCV library.

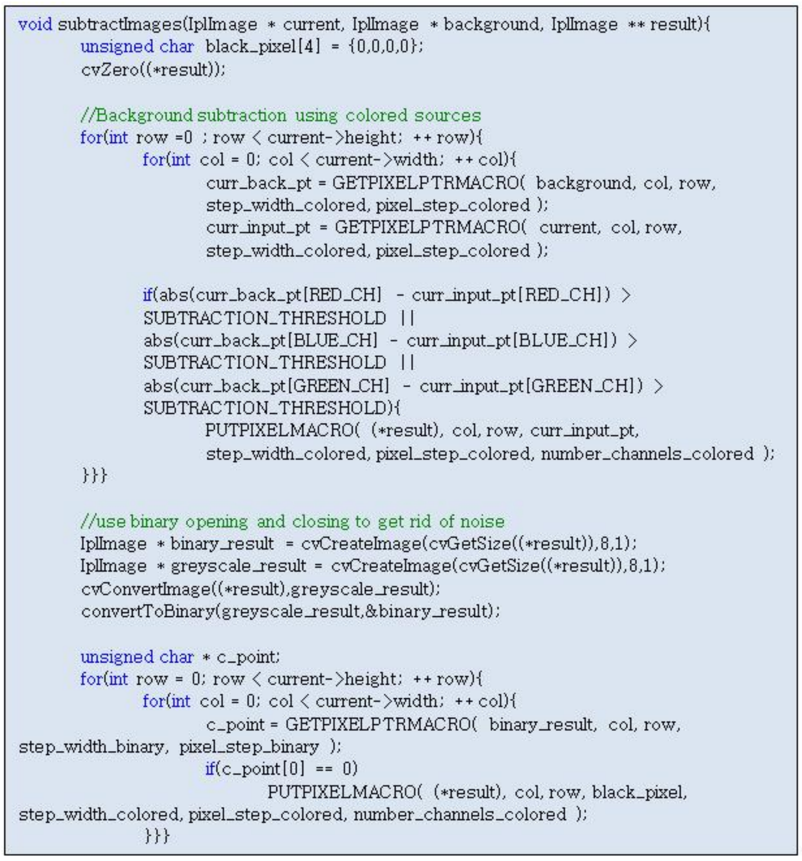


```
public class ClientReplies extends Thread{
      @Override
      public void run(){
            while(true){
                  for(int i = 1; i < 9; i++){
                        if(t[i]!= null){
                            if(t[i].getTargetStatus()){
                  target_status_camera[t[i].getClientID()] = true;
            map_img.setDirectionAngle(t[i].getDirectionAngle() +
            angle_shift[i]);
            }
            else
                  target_status_camera[t[i].getClientID()] = false;
                  }
            }

//indicates whether any of the cameras have detected the target
      if(target_status_camera[1] || target_status_camera[2] ||
      target_status_camera[3] || target_status_camera[4] ||
      target_status_camera[5] || target_status_camera[6] ||
      target_status_camera[7] || target_status_camera[8]){

      if((target_status_camera[1] || target_status_camera[6]) &&
      !target_status_camera[5]){
            map_img.setTargetLocation(POSITION[1]);
            }

      else if(target_status_camera[3] && !target_status_camera[2] &&
      !target_status_camera[6])

      map_img.setTargetLocation(POSITION[3]);

      else if(target_status_camera[4] && !target_status_camera[2])

      map_img.setTargetLocation(POSITION[0]);

      else if(target_status_camera[1] || target_status_camera[2] ||
      target_status_camera[3] || target_status_camera[5] ||
      target_status_camera[6] || target_status_camera[7] ||
      target_status_camera[8])

      map_img.setTargetLocation(POSITION[2]);

      map_img.setTargetStatus(true);
      }
      else{
            map_img.setTargetStatus(false);
            counter = 0;
      }     }     }}
```

Figure 4.9: Client Responses Correlation at The Base Station.

Difference Image Calculator

Figure 4.10 shows how the difference image is calculated by subtracting the pixel values at similar positions in the current frame and the background image. The result image will contain the current frame pixel values where the difference is greater than a defined threshold or zero if otherwise.

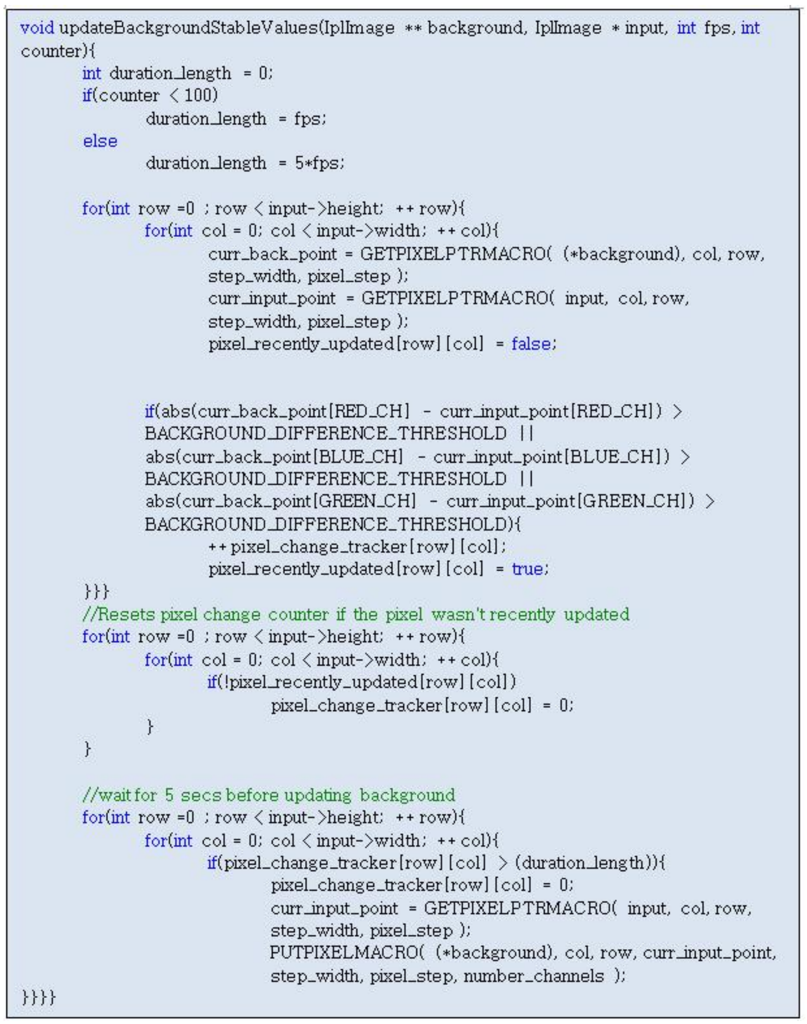


```
void subtractImages(IplImage * current, IplImage * background, IplImage ** result){
        unsigned char black_pixel[4] = {0,0,0,0};
        cvZero((*result));

        //Background subtraction using colored sources
        for(int row =0 ; row < current->height; ++row){
                for(int col = 0; col < current->width; ++col){
                        curr_back_pt = GETPIXELPTRMACRO( background, col, row,
                        step_width_colored, pixel_step_colored );
                        curr_input_pt = GETPIXELPTRMACRO( current, col, row,
                        step_width_colored, pixel_step_colored );

                if(abs(curr_back_pt[RED_CH] - curr_input_pt[RED_CH]) >
                SUBTRACTION_THRESHOLD ||
                abs(curr_back_pt[BLUE_CH] - curr_input_pt[BLUE_CH]) >
                SUBTRACTION_THRESHOLD ||
                abs(curr_back_pt[GREEN_CH] - curr_input_pt[GREEN_CH]) >
                SUBTRACTION_THRESHOLD){
                        PUTPIXELMACRO( (*result), col, row, curr_input_pt,
                        step_width_colored, pixel_step_colored, number_channels_colored );
        }}}

        //use binary opening and closing to get rid of noise
        IplImage * binary_result = cvCreateImage(cvGetSize((*result)),8,1);
        IplImage * greyscale_result = cvCreateImage(cvGetSize((*result)),8,1);
        cvConvertImage((*result),greyscale_result);
        convertToBinary(greyscale_result,&binary_result);

        unsigned char * c_point;
        for(int row = 0; row < current->height; ++row){
                for(int col = 0; col < current->width; ++col){
                        c_point = GETPIXELPTRMACRO( binary_result, col, row,
step_width_binary, pixel_step_binary );
                        if(c_point[0] == 0)
                                PUTPIXELMACRO( (*result), col, row, black_pixel,
step_width_colored, pixel_step_colored, number_channels_colored );
                }}}
```

Figure 4.10: Difference Image Calculator Implementation.

Background Estimation Using Stable Values

The implementation of the Stable Values algorithm used for calculating the background image is discussed in more details in Section 3.4.2.2. Figure 4.11 shows how the algorithm can be implemented using the OpenCV library.

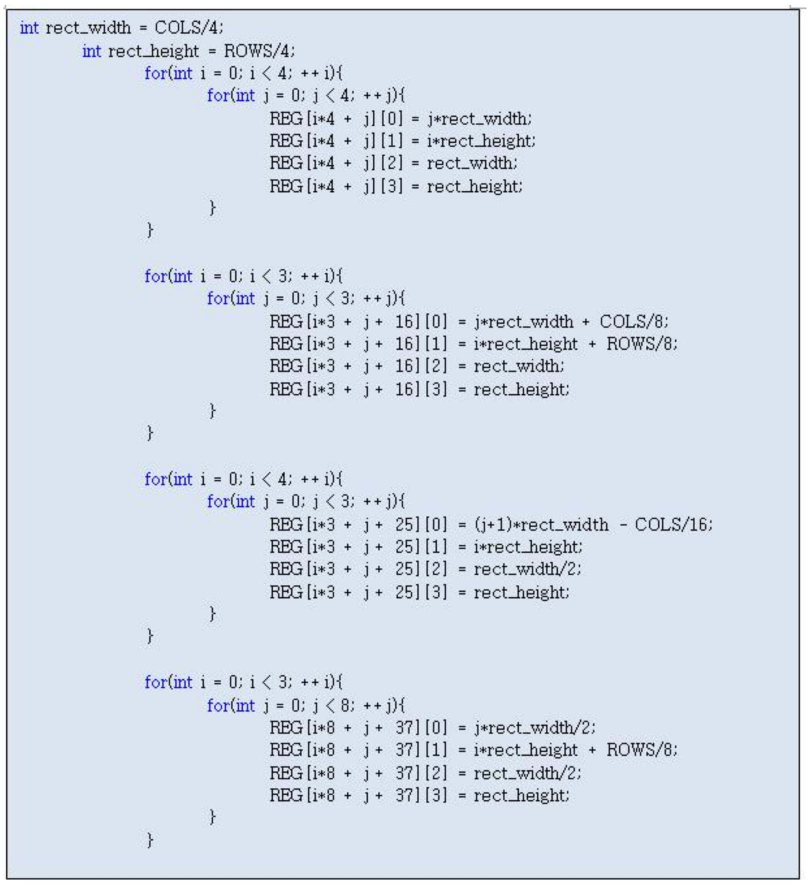


```
void updateBackgroundStableValues(IplImage ** background, IplImage * input, int fps, int
counter){
        int duration_length = 0;
        if(counter < 100)
                duration_length = fps;
        else
                duration_length = 5*fps;

        for(int row =0 ; row < input->height; ++row){
                for(int col = 0; col < input->width; ++col){
                        curr_back_point = GETPIXELPTRMACRO( (*background), col, row,
                        step_width, pixel_step );
                        curr_input_point = GETPIXELPTRMACRO( input, col, row,
                        step_width, pixel_step );
                        pixel_recently_updated[row] [col] = false;


                if(abs(curr_back_point[RED_CH] - curr_input_point[RED_CH]) >
                BACKGROUND_DIFFERENCE_THRESHOLD ||
                abs(curr_back_point[BLUE_CH] - curr_input_point[BLUE_CH]) >
                BACKGROUND_DIFFERENCE_THRESHOLD ||
                abs(curr_back_point[GREEN_CH] - curr_input_point[GREEN_CH]) >
                BACKGROUND_DIFFERENCE_THRESHOLD){
                        ++pixel_change_tracker[row] [col];
                        pixel_recently_updated[row] [col] = true;
        }}}
        //Resets pixel change counter if the pixel wasn't recently updated
        for(int row =0 ; row < input->height; ++row){
                for(int col = 0; col < input->width; ++col){
                        if(!pixel_recently_updated[row] [col])
                                pixel_change_tracker[row] [col] = 0;
                }
        }

        //wait for 5 secs before updating background
        for(int row =0 ; row < input->height; ++row){
                for(int col = 0; col < input->width; ++col){
                        if(pixel_change_tracker[row] [col] > (duration_length)){
                                pixel_change_tracker[row] [col] = 0;
                                curr_input_point = GETPIXELPTRMACRO( input, col, row,
                                step_width, pixel_step );
                                PUTPIXELMACRO( (*background), col, row, curr_input_point,
                                step_width, pixel_step, number_channels );
}}}}
```

Figure 4.11: Background Estimation Using Stable Values Implementation.

Frame Segmentation

Frame segmentation, discussed in more details in Section 3.4.2.4., is an automated process applied to every frame. The algorithm takes into consideration the size of the frame and segments accordingly. Figure 4.12 shows how this is implemented while Figure 3.9 shows the end result.

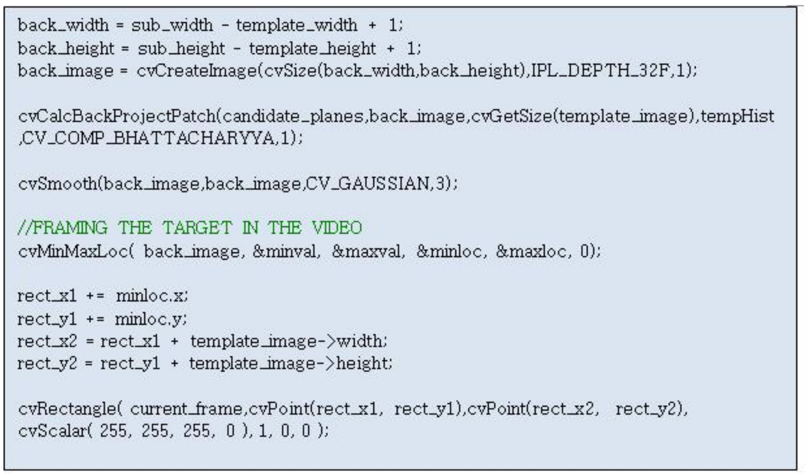


```
int rect_width = COLS/4;
        int rect_height = ROWS/4;
                for(int i = 0; i < 4; ++i){
                        for(int j = 0; j < 4; ++j){
                                REG[i*4 + j][0] = j*rect_width;
                                REG[i*4 + j][1] = i*rect_height;
                                REG[i*4 + j][2] = rect_width;
                                REG[i*4 + j][3] = rect_height;
                        }
                }

                for(int i = 0; i < 3; ++i){
                        for(int j = 0; j < 3; ++j){
                                REG[i*3 + j + 16][0] = j*rect_width + COLS/8;
                                REG[i*3 + j + 16][1] = i*rect_height + ROWS/8;
                                REG[i*3 + j + 16][2] = rect_width;
                                REG[i*3 + j + 16][3] = rect_height;
                        }
                }

                for(int i = 0; i < 4; ++i){
                        for(int j = 0; j < 3; ++j){
                                REG[i*3 + j + 25][0] = (j+1)*rect_width - COLS/16;
                                REG[i*3 + j + 25][1] = i*rect_height;
                                REG[i*3 + j + 25][2] = rect_width/2;
                                REG[i*3 + j + 25][3] = rect_height;
                        }
                }

                for(int i = 0; i < 3; ++i){
                        for(int j = 0; j < 8; ++j){
                                REG[i*8 + j + 37][0] = j*rect_width/2;
                                REG[i*8 + j + 37][1] = i*rect_height + ROWS/8;
                                REG[i*8 + j + 37][2] = rect_width/2;
                                REG[i*8 + j + 37][3] = rect_height;
                        }
                }
```

Figure 4.12: Frame Segmentation Implementation.

Object Detection using Histogram Back Projection

Figure 4.13 illustrates how HBP was implemented in OpenCV. The values resulting from comparing two histograms at each point are stored in an image that has a size equal to the difference between the target image and the template image plus 1.

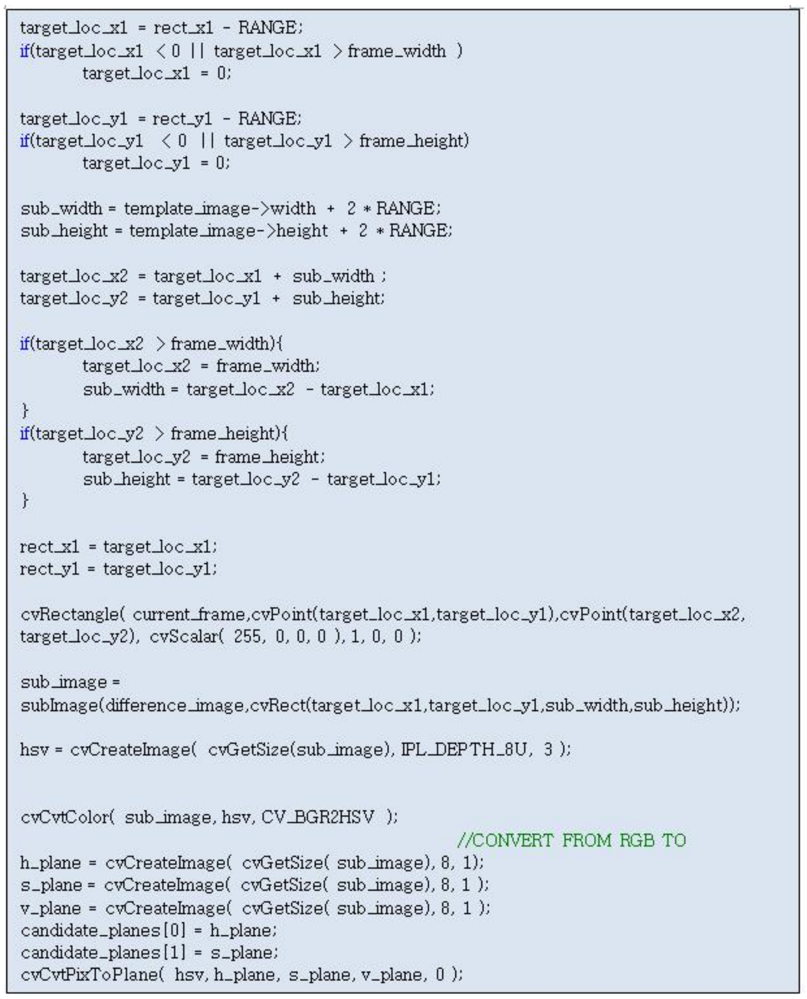


```
back_width = sub_width - template_width + 1;
back_height = sub_height - template_height + 1;
back_image = cvCreateImage(cvSize(back_width,back_height),IPL_DEPTH_32F,1);

cvCalcBackProjectPatch(candidate_planes,back_image,cvGetSize(template_image),tempHist
,CV_COMP_BHATTACHARYYA,1);

cvSmooth(back_image,back_image,CV_GAUSSIAN,3);

//FRAMING THE TARGET IN THE VIDEO
cvMinMaxLoc( back_image, &minval, &maxval, &minloc, &maxloc, 0);

rect_x1 += minloc.x;
rect_y1 += minloc.y;
rect_x2 = rect_x1 + template_image->width;
rect_y2 = rect_y1 + template_image->height;

cvRectangle( current_frame,cvPoint(rect_x1, rect_y1),cvPoint(rect_x2, rect_y2),
cvScalar( 255, 255, 255, 0 ), 1, 0, 0 );
```

Figure 4.13: HBP Implementation.

Object Tracking using Kalman Filter

For this system, the Kalman filter allows the program to narrow down the region of interest while searching for the target in new frames. Searching for the target will be limited to the vicinity of the previous location the target was detected at. Figure 4.14 illustrates how this was implemented.

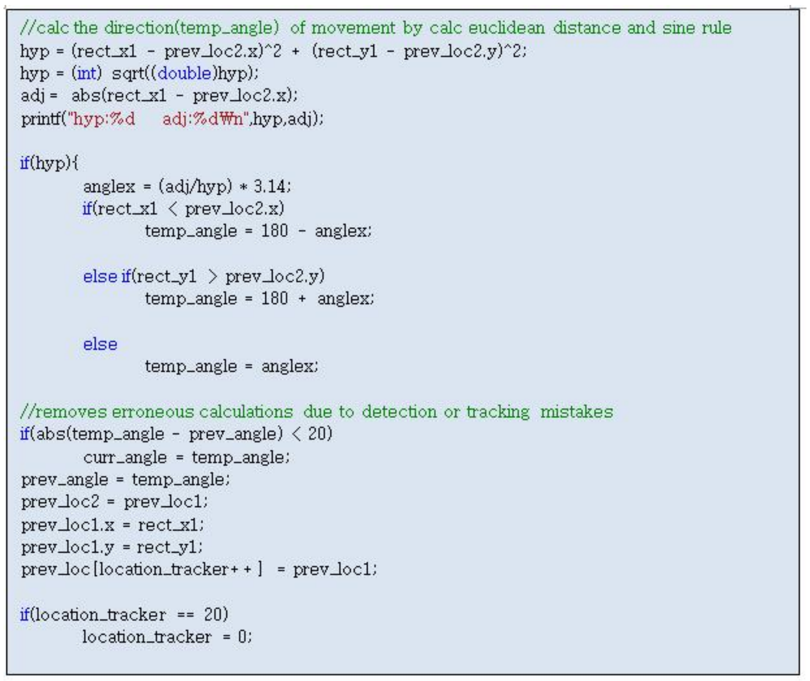


```
target_loc_x1 = rect_x1 - RANGE;
if(target_loc_x1  < 0 || target_loc_x1  > frame_width )
        target_loc_x1 = 0;

target_loc_y1 = rect_y1 - RANGE;
if(target_loc_y1  < 0  || target_loc_y1  > frame_height)
        target_loc_y1 = 0;

sub_width = template_image->width  +  2 * RANGE;
sub_height = template_image->height  +  2 * RANGE;

target_loc_x2 = target_loc_x1  +  sub_width ;
target_loc_y2 = target_loc_y1  +  sub_height;

if(target_loc_x2  > frame_width){
        target_loc_x2 = frame_width;
        sub_width = target_loc_x2  - target_loc_x1;
}
if(target_loc_y2  > frame_height){
        target_loc_y2 = frame_height;
        sub_height = target_loc_y2  - target_loc_y1;
}

rect_x1 = target_loc_x1;
rect_y1 = target_loc_y1;

cvRectangle( current_frame,cvPoint(target_loc_x1,target_loc_y1),cvPoint(target_loc_x2,
target_loc_y2), cvScalar( 255, 0, 0, 0 ), 1, 0, 0 );

sub_image =
subImage(difference_image,cvRect(target_loc_x1,target_loc_y1,sub_width,sub_height));

hsv = cvCreateImage(  cvGetSize(sub_image), IPL_DEPTH_8U,  3 );


cvCvtColor(  sub_image, hsv, CV_BGR2HSV  );
                                                //CONVERT FROM RGB TO
h_plane = cvCreateImage(  cvGetSize( sub_image), 8, 1);
s_plane = cvCreateImage(  cvGetSize(  sub_image), 8, 1 );
v_plane = cvCreateImage(  cvGetSize(  sub_image), 8, 1 );
candidate_planes[0] = h_plane;
candidate_planes[1] = s_plane;
cvCvtPixToPlane(  hsv, h_plane,  s_plane, v_plane,  0 );
```

Figure 4.14: Kalman Filter Implementation.

Direction of Movement Determination

The direction of movement algorithm allows the client camera node to estimate the direction the target is moving in based on the current and previous locations the target is/was detected at. This allows the tracking process to be executed in a smarter and more efficient manner by limiting the search process to cameras located around that direction. It as well gives the user at the base station an idea in which direction the target is moving in. This equation used has been discussed in Section 3.4.2.8. while Figure 4.15 shows how it was implemented.



```
//calc the direction(temp_angle) of movement by calc euclidean distance and sine rule
hyp = (rect_x1 - prev_loc2.x)^2 + (rect_y1 - prev_loc2.y)^2;
hyp = (int) sqrt((double)hyp);
adj = abs(rect_x1 - prev_loc2.x);
printf("hyp:%d    adj:%d₩n",hyp,adj);

if(hyp){
        anglex = (adj/hyp) * 3.14;
        if(rect_x1 < prev_loc2.x)
                temp_angle = 180 - anglex;

        else if(rect_y1 > prev_loc2.y)
                temp_angle = 180 + anglex;

        else
                temp_angle = anglex;

//removes erroneous calculations due to detection or tracking mistakes
if(abs(temp_angle - prev_angle) < 20)
        curr_angle = temp_angle;
prev_angle = temp_angle;
prev_loc2 = prev_loc1;
prev_loc1.x = rect_x1;
prev_loc1.y = rect_y1;
prev_loc[location_tracker++] = prev_loc1;

if(location_tracker == 20)
        location_tracker = 0;
```

Figure 4.15: Direction of Movement Implementation.

4.3.2 Middleware Component

Since the computer vision component runs separately from the middleware component, it is required to have some communication between the two that will allow the middleware component to transmit to the base station what the computer vision component has processed. For this, a text file has been used to which one writes and the reads from, the computer vision application writes while the middleware application reads. Any read and writes exceptions are handled in a way the prevents any errors. Since most of the data to be read will not undergo any sudden change, they both read and write every one second. The visions application does this by counting the number of frames and comparing it to the number of frames per second the video is being played at, while the middleware application implements this by causing the accessing thread to sleep for a second. Random File Access in Appendix F, shows how this was implemented.

Chapter 5

Evaluation

For this project, a prototype of the system described in Chapters 3 and 4 was developed. In this Chapter the developed prototype and the algorithms used will be evaluated. The performance of the prototype will be evaluated, as well as compared against the original system for scalability purposes. The major differences between the two is mostly related to scale, the topology used, processing power at the client nodes, the footage collected at the client nodes, and the actual network bandwidth.

The chapter discusses the efficiency of the algorithms used for implementing each component, and whether the use of other algorithms may improve the performance of the system. Integrating additional algorithms to the ones implemented is also considered, as the combination of two or more algorithms for the execution of a certain task is more likely to yield better results. In addition, the chapter evaluates design choices and their suitability for their parent components and the system as a whole.

5.1 Mobile Application

The mobile application built for this system, was designed to run on an Android mobile phone. Testing of this application was applied at several stages through out the development phase, using white and black box testing methods. Each component comprising this application was tested individually once built, then again once integrated into the whole system. The testing cases checked the components for errors, bugs and whether it executed as expected. Other tests involved stress testing which allows the evaluation of the robustness of the application. The test results indicated the application's performance is within tolerance.

Aside from the test cases, development for the android platform was associated with several bugs related to the Android OS. This hindered the development process and required the alteration of the methods use. However, the android support team were responsive and addressed issues as soon as possible which was helpful in some cases.

5.2 The Base Station

The base station built for this prototype did not undergo as many tests as the other system components. This is due to the fact that the server built was designed specifically for this prototype, thus to support a limited number of client camera nodes. Even though the original system was taken into consideration while building the base station in this model, the base station was optimized to suit the addressed scenario. The number of clients that can connect was limited to 8, which is the number of client camera nodes available from the PETS 2010 workshop [24]. However, the methods implemented and their execution was rigorously tested.

The server was designed to accept both TCP and UDP requests. Upon testing, the dual protocol support proved advantageous to the system as each was found suitable for the type of communication it was implemented for. For instance, the UDP protocol suffers from several demerits related to the protocol's lack of any error handling mechanisms, no packet lost support, possible out of order delivery or even duplicate packet delivery. However, for this system UDP was used for the transmission of small packages, usually one datagram, containing the image query from the mobile application to the server. In this case UDP removes all the time delays required to establish and tear down connections that may have affected the system's performance negatively if it was based on TCP.

As for the GUI, the design took into consideration visual clarity as much as possible. Since such an application is aimed for security purposes, keeping the design as simple, intuitive and clear was the goal. The user was presented with a map such as that in Figure 3.6 on which the information was displayed. The interface presents the user with updates regarding the detected target, in a way that will grab his/her refraining from the use of any visual distractions. However, the application does not allow the user to have any kind of control over the camera nodes or to send any feedback regarding the detected target. Such a feedback may assist in removing any false positives identified by the camera nodes, thus providing a more accurate tracking and detection service.

Overall, there server application designed and developed for the base station performed quite well, however, it was tested for a limited number of clients in a predefined and limited geographical area. Whether the system scales well to perform similarly for a larger number of client cameras is not known.

5.3 Client Camera Node

Most of the detection and tracking process occurs at the client camera nodes. The camera nodes connect directly to the base station and exchange information that may be interpreted at both ends based on a protocol specifically designed for this system. The protocol was designed to be implemented on top the TCP protocol, therefore, taking advantage of the quality assurance tools provided by the protocol.

In relation to the computer vision aspect, the detection and tracking algorithms implemented were evaluated with respect to their performance and computational cost. As a result of running several tests, it was observed that the target detection rate in the tested scenarios varied from one camera to another. The highest observed detection rate was around 87% in camera views that captured the target from a similar angle to that in the template image. While the lowest detection rate was around 35% in camera views that captured the target from a different angle view and had different birghtness levels than these in the template image. It was also noticeable that the system did not detect any false positives, which proves the importance and the effectiveness of the applied filters, especially these related to size and the speed of movement. The detection rate was estimated by measuring how many times the target was correctly identified in a single sighting from the time it enters the FoV of that camera till it exits it. The detection rate when using HBP was noticeably higher than that from using template matching, where the highest was at 40%Ṫhus supporting the decision of replacing template matching with HBP as the implemented algorithm for object detection. However, from the cost of computation perspective, HBP was found to be quite expensive. During early implementations of the HBP method, especially the Local Histogramming HBP method, it took an average of 1 minute and 20 seconds to process a single frame. The machine used to run the application on was equipped with an 2.4GHz Intel Core2Quad CPU , 3GB of RAM and an NVIDIA GeForce 8800 GTX GPU. This cost was later reduced to allow the application to be used for processing videos running at 15 fps. This was achieved through the integration of segmentation, size thresholds, and a kalman filter with the HBP method. Through the integration of the latter techniques, the application is capable of detecting and tracking targets in videos that run at 15 fps without causing any delays.

5.4 Overall System Evaluation

Based on the different tests that have been applied to evaluate the system's components individually and as a whole, in a scenario acquired from the PETS 2010 vision workshops database [24], the built prototype was found to meet the proposed requirements in Section 1.3. The test results prove the robustness of the system and its ability to detect and track a target based on an image query created using a smart phone application. The system was designed in a way that took scalability and extensibility into consideration. However, none of the test cases evaluated the system's performance and robustness in a scenario where it was required to support a large number of client camera nodes.

Chapter 6

Conclusion

The prototype designed and developed for this project as an initial demonstration of a collaborative video surveillance system shows how a target may be detected and tracked in the FoVs of several client camera nodes. The detection and tracking engine at the client node uses a smart phone created image query as the template image. The system was developed using the incremental and iterative development model which allows the building of robust components that were heavily tested individually and as a whole, using both black box and white box testing. Thus, creating a system that performs within expectations upon integration of the separate components. This, as well, assisted in developing for components that are different in their application and the platforms they are built on. The system lacked certain features that will be discussed in more details in Section 6.1 covering the future work aspect of this report, however all the main requirements were met.

As discussed in Section 2.1, the requirements for this system were:

- To design a network that allows the exchange of information between a mobile phone, a base station and several client camera nodes.
- To design and develop a mobile application that allows a user to create an image query. The application should allow the user to capture, edit and send images using

the mobile phone.

- To design and develop a smart camera application that is capable of detecting and tracking targets based on the received image query.
- To visually present correlated data received from different client cameras at the base station.
- To efficiently detect and track a target from the field of view of one camera to that of another.

The network designed and developed for this system allows the base station to receive an image query from a smart phone and distribute it to the client camera nodes. The image query is received as a datagram using the UDP protocol while the TCP protocol was used to establish connections between the base station and the client camera nodes.

A mobile application was designed and developed for the Android platform. The application after installation allows its user to capture, edit and send a photo to the base station using any smart phone running the Android OS. The editing feature allows the user to specify the exact location of the target within the image, thus creating an image query. The image query will be used by the camera nodes as a template image to detect and track the specified target.

The client camera node was designed as two separate applications that collaborate in creating on of the system's components. Each application was designed as an individual element using different programming languages. Both applications run simultaneously and collaborate in providing a single service. One application is the middleware component of the client camera node while the other is related to the computer vision aspect of it. Using the developed middleware layer, the computer vision application receives and transmits information related to the target that is to be or being tracked. A combination of computer vision algorithms that execute in a sequential order allow the detection and tracking of the target.

A multithreaded server that handles both UDP and TCP requests while providing up-to-date visual feedback was built to act as the system's base station. Besides allowing the exchange of information between the different system components, the base station visually presents to the user information received from the client camera nodes. Such information is mostly related to the target being tracked, his/her location and the direction he/she may be moving in. Information received from various camera nodes is correlated at the base station in order to provide the most accurate target related information.

Overall, the system allows the user to detect and track a target from the FoV of one camera node to another while utilizing the available bandwidth and the processing power at the node. This was achieved by triggering the detection and tracking process only at relevant nodes, depending on previous sightings of the target from either the mobile phone application or by other camera nodes.

6.1 Future Work

The scope of this project inspires many ideas and makes room for several inquiries that makes the potential of future growth inevitable. Future development may be applied to the system as a whole and to individual components as well. Starting with the mobile application, developing multiple versions of the application that runs on a number of mobile platforms such as Windows Mobile 7, Blackberry and iOS. In addition, the application should be further improved to include features that will provide the user with more powerful editing tools. Such a feature provides better accuracy to the user while cropping, thus excluding from the image query any background related information, that is irrelevant to the target being tracked. Besides that, the fact that the mobile application user is the only user that might have seen the target in real life, it would be beneficial to the system if he/she were able to give either positive or negative feedback on the candidate targets detected by the camera nodes.

As for the base station, it should allow the user to give feedback on the candidate targets detected by the camera nodes, in a similar fashion to that discussed for the mobile application. The base station may as well take the system's scalability into consideration. The base station built for this prototype, is capable of establishing connections to a limited number of client camera nodes. This may be altered in a way that will allow the system to be more scalable and extensible, while retaining it's robustness and performance levels.

Last but not least, the client camera nodes. The computer vision techniques used at the client camera nodes accomplish the required tasks in the test scenarios, however the algorithms performance wasn't constantly within accepted tolerance range. Therefore, different algorithms, such as feature based object detection, may be used in addition to HBP. This will improve the systems performance by boosting the true positives detection rate, yet decrease the number of false positives. In addition, camera nodes may be upgraded to include capabilities that allow the detection and tracking of multiple targets that may simultaneously co-exist in its FoV. Taking into consideration the regular advancements in visual equipment, future smart camera nodes are quite likely to be equipped with visual sensors capable of recording high quality video. The algorithms to be used in the detection process may as well take this into consideration.

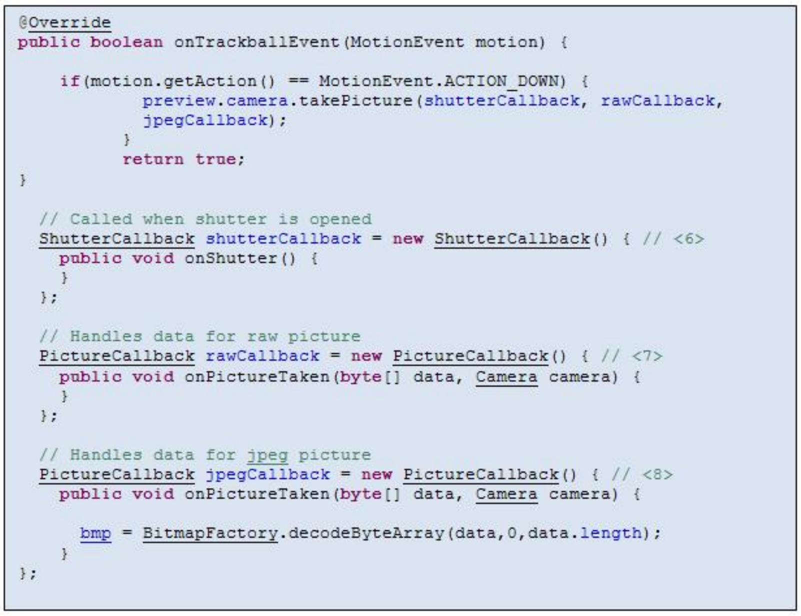
Appendix A

Abbreviations

Short Term	**Expanded Term**
CIBR	Content-Based Image Retrieval
DSC	Distributed Smart Cameras
FoV	Field of View
FPS	Frames per Second
HBP	Histogram Back Projection
LEACH	Low Energy Adaptive Cluster Hierarchy
NTP	Network Time Protocol
OSI	Open System Interconnection
PEGASIS	Power Efficient Gathering in Sensor Information System
PSC	Pervasive Smart Cameras
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VSN	Visual Sensor Network

Appendix B

Android Photo Activity



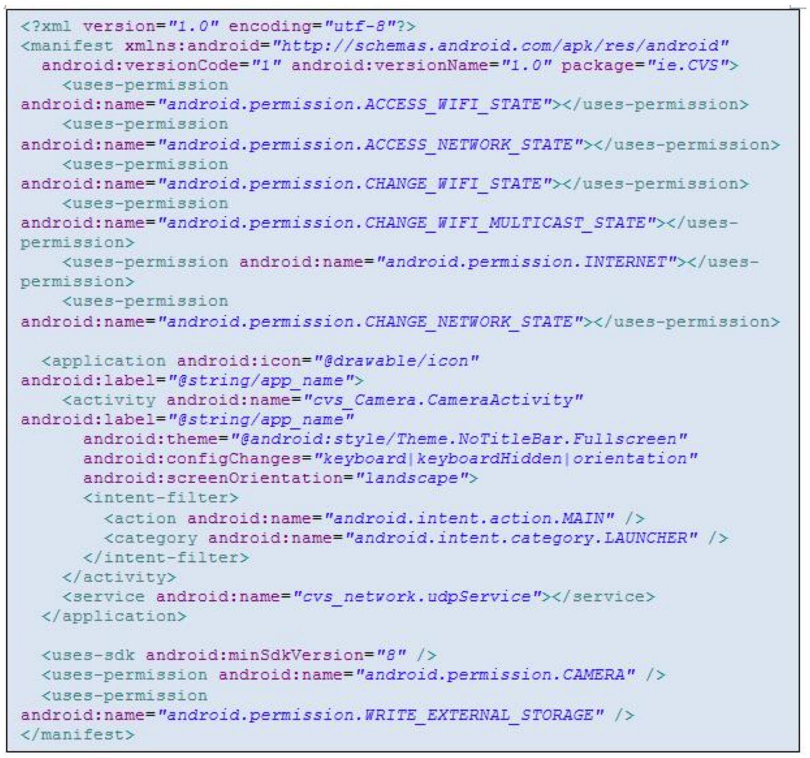
```
@Override
public boolean onTrackballEvent(MotionEvent motion) {

    if(motion.getAction() == MotionEvent.ACTION_DOWN) {
            preview.camera.takePicture(shutterCallback, rawCallback,
            jpegCallback);
          }
          return true;
}

  // Called when shutter is opened
  ShutterCallback shutterCallback = new ShutterCallback() { // <6>
    public void onShutter() {
    }
  };

  // Handles data for raw picture
  PictureCallback rawCallback = new PictureCallback() { // <7>
    public void onPictureTaken(byte[] data, Camera camera) {
    }
  };

  // Handles data for jpeg picture
  PictureCallback jpegCallback = new PictureCallback() { // <8>
    public void onPictureTaken(byte[] data, Camera camera) {

      bmp = BitmapFactory.decodeByteArray(data,0,data.length);
    }
};
```

Figure B.1: Photo Capturing Methods.

The code in Figure B.1 describes how the application listens to events on the trackball button and captures a photo once the button is pressed. A push on the button will provoke calls to the shutter, data collection and jpeg classes. These classes gather the data read by the device's lens and compresses the data into JPG format. It's worth mentioning that the JPG compression is only used to store a local copy of the image. However, prior to transmitting the image to the base station, the data is compressed into PNG format, which is lossless, this gives more information to the camera nodes to compare against when searching for the target.

Appendix C

Android Manifest



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  android:versionCode="1" android:versionName="1.0" package="ie.CVS">
    <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
    <uses-permission
android:name="android.permission.CHANGE_WIFI_STATE"></uses-permission>
    <uses-permission
android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE"></uses-
permission>
    <uses-permission android:name="android.permission.INTERNET"></uses-
permission>
    <uses-permission
android:name="android.permission.CHANGE_NETWORK_STATE"></uses-permission>

  <application android:icon="@drawable/icon"
android:label="@string/app_name">
    <activity android:name="cvs_Camera.CameraActivity"
android:label="@string/app_name"
      android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
      android:configChanges="keyboard|keyboardHidden|orientation"
      android:screenOrientation="landscape">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <service android:name="cvs_network.udpService"></service>
  </application>

  <uses-sdk android:minSdkVersion="8" />
  <uses-permission android:name="android.permission.CAMERA" />
  <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```

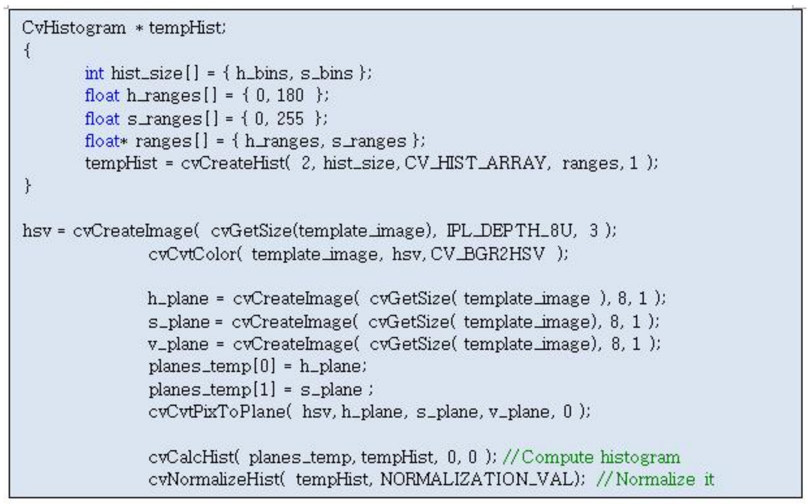
Figure C.1: Defining Permissions in Android Manifest.

The code in Figure C.1 shows the code stored in the applications manifest. It shows all the activities, services and android components that have been extended in this application. It as well defines the themes each of these components will use and how they will be displayed. Besides that, the manifest defines access permissions that are given to this application. The access permissions are displayed to the applications user upon installation of the application, which is one of android's distinguishable security features.

Appendix D

Histograms

Figure D.1 illustrates how histograms are calculated for each image in OpenCV and the data structures used for this purpose.



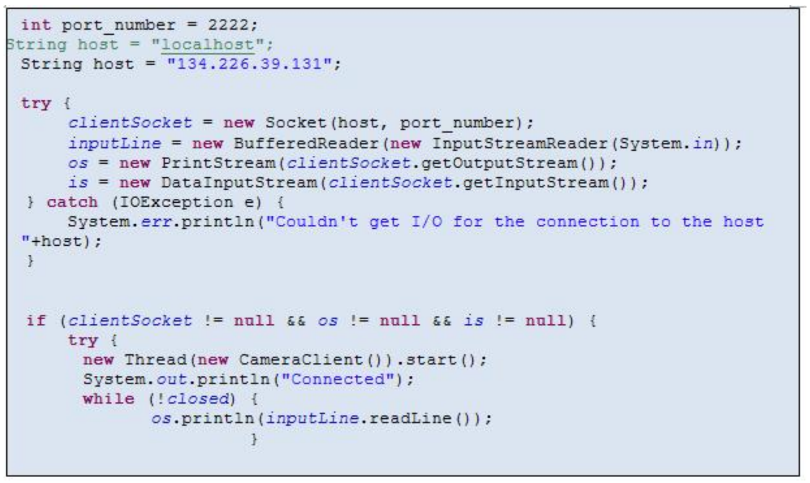
```
CvHistogram * tempHist;
{
        int hist_size[] = { h_bins, s_bins };
        float h_ranges[] = { 0, 180 };
        float s_ranges[] = { 0, 255 };
        float* ranges[] = { h_ranges, s_ranges };
        tempHist = cvCreateHist( 2, hist_size, CV_HIST_ARRAY, ranges, 1 );
}

hsv = cvCreateImage( cvGetSize(template_image), IPL_DEPTH_8U, 3 );
                cvCvtColor( template_image, hsv, CV_BGR2HSV );

                h_plane = cvCreateImage( cvGetSize( template_image ), 8, 1 );
                s_plane = cvCreateImage( cvGetSize( template_image), 8, 1 );
                v_plane = cvCreateImage( cvGetSize( template_image), 8, 1 );
                planes_temp[0] = h_plane;
                planes_temp[1] = s_plane ;
                cvCvtPixToPlane( hsv, h_plane, s_plane, v_plane, 0 );

                cvCalcHist( planes_temp, tempHist, 0, 0 ); //Compute histogram
                cvNormalizeHist( tempHist, NORMALIZATION_VAL); //Normalize it
```

Figure D.1: Histograms Implementation.

Appendix E

Sockets

The code in Figure E.1 shows how the connection is established between a client camera node and the base station. It defines the port and the IP address of the base station, then creates a thread that handles all incoming and outgoing requests on the created socket.



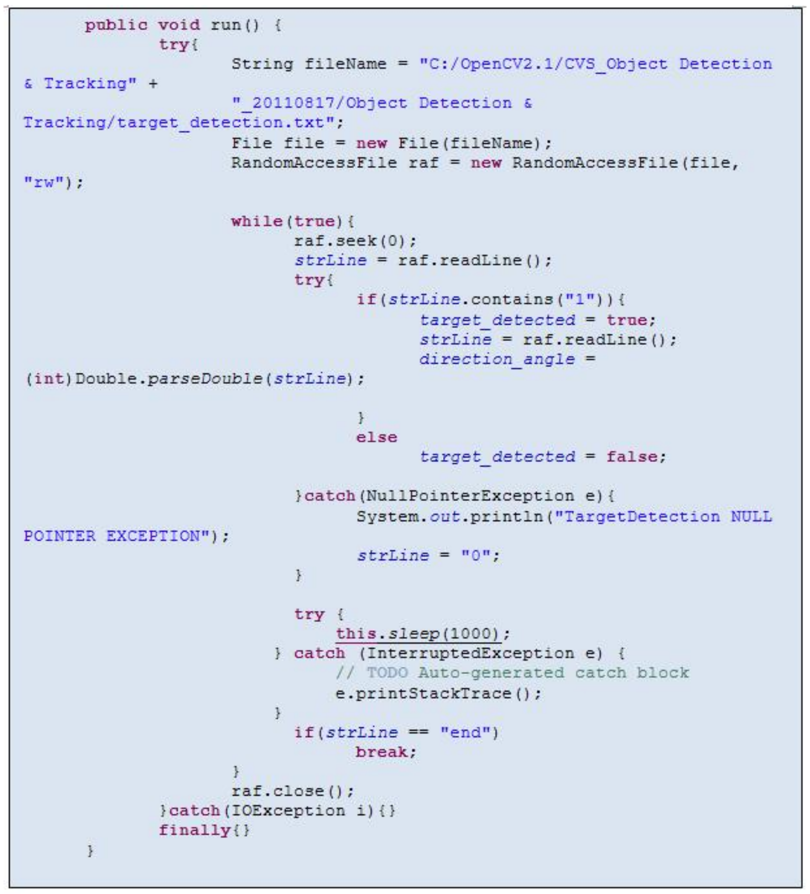
```
 int port_number = 2222;
String host = "localhost";
 String host = "134.226.39.131";

 try {
      clientSocket = new Socket(host, port_number);
      inputLine = new BufferedReader(new InputStreamReader(System.in));
      os = new PrintStream(clientSocket.getOutputStream());
      is = new DataInputStream(clientSocket.getInputStream());
  } catch (IOException e) {
      System.err.println("Couldn't get I/O for the connection to the host
 "+host);
  }


  if (clientSocket != null && os != null && is != null) {
      try {
        new Thread(new CameraClient()).start();
        System.out.println("Connected");
        while (!closed) {
                os.println(inputLine.readLine());
                      }
```

Figure E.1: Sockets Implementation.

Appendix F

Random File Access

Figure F.1 shows the middleware code that reads from the text file at the client camera node. This allows the vision application to communicate with network application that run in parallel at the client camera node.



```
	public void run() {
		try{
			String fileName = "C:/OpenCV2.1/CVS_Object Detection
& Tracking" +
			"_20110817/Object Detection &
Tracking/target_detection.txt";
			File file = new File(fileName);
			RandomAccessFile raf = new RandomAccessFile(file,
"rw");

			while(true){
				raf.seek(0);
				strLine = raf.readLine();
				try{
					if(strLine.contains("1")){
						target_detected = true;
						strLine = raf.readLine();
						direction_angle =
(int)Double.parseDouble(strLine);

					}
					else
						target_detected = false;

				}catch(NullPointerException e){
					System.out.println("TargetDetection NULL
POINTER EXCEPTION");
					strLine = "0";
				}

				try {
					this.sleep(1000);
				} catch (InterruptedException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
				if(strLine == "end")
					break;
			}
			raf.close();
		}catch(IOException i){}
		finally{}
	}
```

Figure F.1: Random File Access.

Bibliography

[1] Saiful Akbar, Josef Küng, and Roland Wagner. Multi-feature integration with relevance feedback on 3d model similarity retrieval. *J. Mob. Multimed.*, 3:235–254, September 2007.

[2] C. Arth, C. Leistner, and H. Bischof. Object reacquisition and tracking in large-scale smart camera networks. In *Distributed Smart Cameras, 2007. ICDSC '07. First ACM/IEEE International Conference on*, pages 156 –163, sept. 2007.

[3] D. Beatty and N. Lopez-Benitez. Image query service using content management techniques. In *Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on*, pages 1335 –1340, april 2009.

[4] WILLIAM M. BULKELEY. Chicago's camera network is everywhere, April 2011.

[5] M. Casares, A. Pinto, Youlu Wang, and S. Velipasalar. Power consumption and performance analysis of object tracking and event detection with wireless embedded smart cameras. In *Signal Processing and Communication Systems, 2009. ICSPCS 2009. 3rd International Conference on*, pages 1 –8, sept. 2009.

[6] Y. Charfi, N. Wakamiya, and M. Murata. Challenging issues in visual sensor networks. *Wireless Communications, IEEE*, 16(2):44 –49, april 2009.

[7] Yongil Cho, Sang Ok Lim, and Hyun Seung Yang. Collaborative occupancy reasoning in visual sensor network for scalable smart video surveillance. *Consumer Electronics, IEEE Transactions on*, 56(3):1997 –2003, aug. 2010.

[8] C.P. Diehl and II Hampshire, J.B. Real-time object classification and novelty detection for collaborative video surveillance. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 3, pages 2620 –2625, 2002.

[9] Weiming Hu, Xue Zhou, Min Hu, and S. Maybank. Occlusion reasoning for tracking multiple people. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19(1):114 –121, jan. 2009.

[10] M. Jovanovic and B. Rinner. Middleware for dynamic reconfiguration in distributed camera systems. In *Intelligent Solutions in Embedded Systems, 2007 Fifth Workshop on*, pages 139 –150, june 2007.

[11] Young-Kee Jung, Kyu-Won Lee, and Yo-Sung Ho. Content-based event retrieval using semantic scene interpretation for automated traffic surveillance. *Intelligent Transportation Systems, IEEE Transactions on*, 2(3):151 –163, sep 2001.

[12] S. Khan, O. Javed, Z. Rasheed, and M. Shah. Human tracking in multiple cameras. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 331 –336 vol.1, 2001.

[13] Chang Hong Lin, W. Wolf, A. Dixon, X. Koutsoukos, and J. Sztipanovits. Design and implementation of ubiquitous smart cameras. In *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on*, volume 1, page 8 pp., june 2006.

[14] Fuqiang Liu, Kai Zhou, and Donglei Wang. Application of video sensor networks in traffic surveillance. In *Sensor and Ad Hoc Communications and Networks, 2006. SECON '06. 2006 3rd Annual IEEE Communications Society on*, volume 3, pages 916 –919, sept. 2006.

[15] Zhifang Liu, Xiaopeng Gao, and Xiang Long. Adaptive random testing of mobile

application. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 2, pages V2–297 –V2–301, april 2010.

[16] M. A. Patricio, J. Carbó, O. Pérez, J. García, and J. M. Molina. Multi-agent framework in visual sensor networks. *EURASIP J. Appl. Signal Process.*, 2007:226–226, January 2007.

[17] B. Rinner, M. Jovanovic, and M. Quaritsch. Embedded middleware on distributed smart cameras. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–1381 –IV–1384, april 2007.

[18] B. Rinner, T. Winkler, W. Schriebl, M. Quaritsch, and W. Wolf. The evolution from single to pervasive smart cameras. In *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, pages 1 –10, sept. 2008.

[19] B. Rinner and W. Wolf. An introduction to distributed smart cameras. *Proceedings of the IEEE*, 96(10):1565 –1575, oct. 2008.

[20] John R. Smith and Shih-Fu Chang. Integrated spatial and feature image query. *Multimedia Syst.*, 7:129–140, March 1999.

[21] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering, 2007.

[22] Stanislava Soro and Wendi Heinzelman. Review article a survey of visual sensor networks.

[23] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition, 2002.

[24] IEEE Computer Society (PAMI TC) and IEEE Signal Processing Society (IVMSP TC). Pets, August 2011.

[25] Android Dev Team. Android guide, August 2011.

[26] Android Dev Team. Android sdk, August 2011.

[27] OpenCV Dev Team. Opencv wiki, August 2011.

[28] A. Murat Tekalp. *Digital video processing.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[29] S. Velipasalar and W. Wolf. Recovering field of view lines by using projective invariants. In *Image Processing, 2004. ICIP '04. 2004 International Conference on*, volume 5, pages 3069 – 3072 Vol. 5, oct. 2004.

[30] Wayne Wolf, Burak Ozer, and Tiehan Lv. Smart cameras as embedded systems. *Computer*, 35:48–53, September 2002.

[31] Min Wu and Chang Wen Chen. Collaborative image coding and transmission over wireless sensor networks. *EURASIP J. Appl. Signal Process.*, 2007:223–223, January 2007.

[32] Ren Yueqing and Xu Lixin. A study on topological characteristics of wireless sensor network based on complex network. In *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, volume 15, pages V15–486 –V15–489, oct. 2010.