Stony Brook University

Automatic Calibration of a Camera Array-based Robot Tracking System

A Thesis Presented

by

GURUSWAMY NAMASIVAYAM

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Computer Science

Stony Brook University

December 2009

Stony Brook University

The Graduate School

Guruswamy Namasivayam

We, the thesis committee for the above candidate for the

Master of Science degree, hereby recommend acceptance of this thesis.

Professor Tzi-cker Chiueh - Thesis Advisor

Department of Computer Science

Professor Samir Das – Defence Committee

Department of Computer Science

Professor Jie Gao – Defence Committee

Department of Computer Science

This thesis is accepted by the Graduate School

Lawrence Martin

Dean of the Graduate School

Abstract of the Thesis

Automatic Calibration of a Camera Array-based Robot Tracking System

by

GURUSWAMY NAMASIVAYAM

Master of Science

in

Computer Science

Stony Brook University

2009

In recent years, there has been an increase in the amount of wireless protocol design and testing activities that depend on wireless testbeds. Most of these testbeds used today are low-cost, miniaturized testbeds and support different forms of mobility. In this thesis we propose improvements to the setup and operations of the Miniaturized wireless Network Testbed, MiNT-m.

The testbed consists of several fully mobile wireless MiNT nodes that are capable of emulating NS scripts. Each mobile node consists of a router board with four wireless cards mounted on commercial off-the-shelf robots. The robots are capable of responding to simple movement commands. The range of the wireless cards are limited by radio frequency attenuation to keep the testbed small. The nodes are identified and located by an array of overhead cameras that scan for differently colored patches attached to the nodes. However, this color based tracking system requires a significant effort in setup time and requires constant maintenance. Also, the current color based scheme does not lend itself to be scaled to hundreds of nodes. Another issue with the current scheme is the precise alignment requirement of the cameras in MiNT testbed. This camera alignment apart from consuming setup time, requires significant overlap between adjacent cameras. The later

requirement results in a smaller testbed than would have been possible with the same number of cameras.

In this thesis, we discuss the problems in the setup that take up considerable time and effort, and propose development of new tools to make setup process significantly faster and more reliable. Also, we suggest significant modifications to the color based node detection algorithm for the MiNT testbed. This would make the testbed easily scale to a large number of nodes, make better utilization of the regions covered by the camera and make the setup process easier. In effect, we propose a complete solution that would make the testbed readily deployable anywhere in the world with minimal time and effort.

To

Almighty God and My Family

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to Thank my advisor Professor. Tzi-cker Chiueh who helped me greatly in the design of the new SIFT based node localization system. He was instrumental in coming up with new ways to solve implementation issues and get the system to work. I would also like to thank Jui–Hao Chiang who helped in setting up the MINT-m testbed, testing the new MINT-m node localization system and in de-bugging issues related to the roomba robot.

Chapter 1

Introduction

1.1 Overview of the MiNT-m Wireless Testbed

A growing body of wireless research with requirement for high fidelity in experiments is moving towards wireless testbeds. In recent years, there has been an increase in the use of miniaturized in-lab testbeds for wireless experiments. However, there are only a few publicly available wireless testbeds available for running experiments. Still, most researchers refrain from setting up their own testbeds as setting up a testbed involves a lot of effort. Typically, most of the testbeds require significant amount of space to setup which in itself is costly, costly hardware and significant manual effort to set up. In this thesis, we try to address the above concerns for the MiNT-m wireless testbed.

MiNT-m wireless testbed uses the off-the-shelf robotic vacuum cleaner, roomba as the robot used to carry a wireless router board with 4 wireless cards. The router board can emulate NS scripts and also issues simple move commands to the roomba robots. The position of the robots is determined by processing the feed from an overhead camera array. The feed from each of the cameras are processed individually in the tracking system. The output from the tracking system is then aggregated in the intermediate server. The system consists of 4 main components,

1. Robot tracking system – The component responsible for locating the node within the testbed.
2. Intermediate server – Component responsible for aggregating the position of all the nodes in the testbed.
3. Controller – component that interfaces with the user and allows the user to run experiments.
4. Node daemon – This process run on each of the router boards and controls the movements of the individual nodes.

The tracking system interfaces with the camera and computes the position and orientation of the image within the camera. This information is then sent to the intermediate server where it gets aggregated and is disseminated to the rest of the system. It then sends this input to the controller. The controller uses this information to plan the trajectory for future destinations and intimates the calculated movement commands to each of the noded processes running in the router boards. These processes then issue the commands to the roomba robot to achieve movement.

1.1.1 MiNT-m Robot Tracking System

MiNT-m uses a color based node identification scheme. The tracking system uses a camera array consisting of six overhead cameras arranged in overlapping fashion. The overlap between each of the cameras must be at least large enough to fit a single MiNT node along any dimension (including the diagonal).

This requirement can be seen in Figure 2. The requirement is enforced to ensure that a MiNT node is covered completely by at least one camera at all times. When a camera moves from one camera to another, say from camera 1 to camera 2, the node is initially covered completely by camera 1, then it moves to the overlap region of camera 1 and 2. During the transition, it is covered by both the cameras, though partially by one of them at some locations. At locations where a node is covered by two cameras, the location of the node as calculated by both the cameras is averaged to produce the node position and orientation.

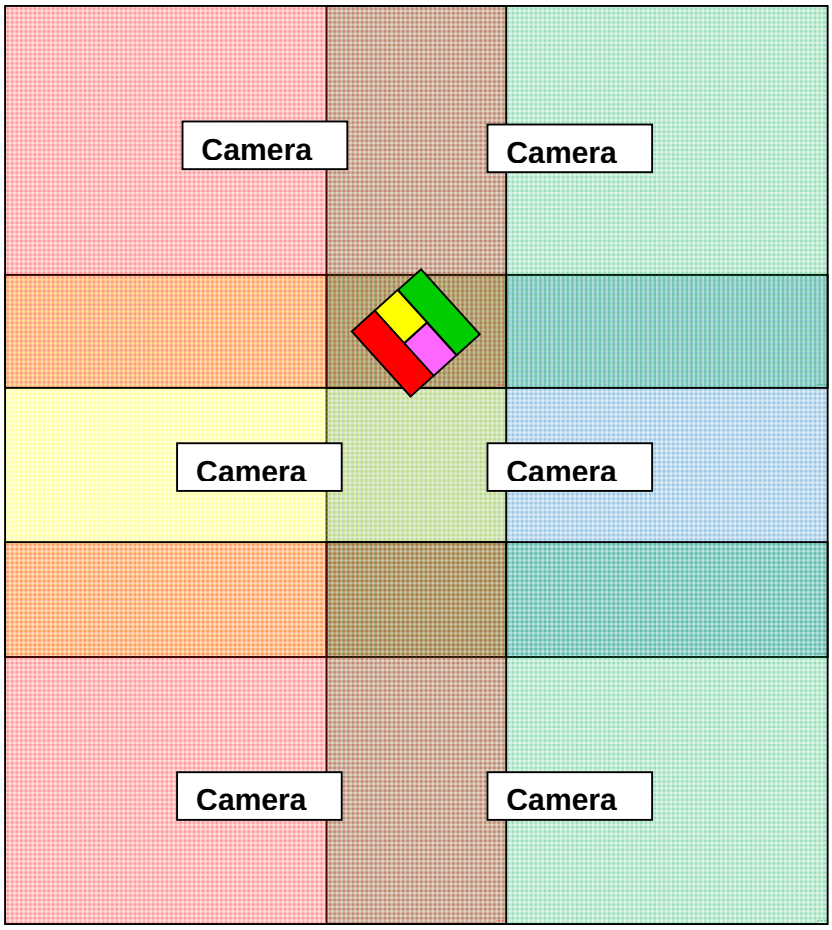

Figure 1 - Schematic diagram of the original MiNT-m testbed showing the degree of camera alignment required and the amount of unusable space due to the mandatory requirement of overlap.

The feed from each camera is processed by a dedicated node detection process. The vision based node detection software was developed by modifying camstream, an open source software that can be used for taking pictures or making videos from webcams. Each of the camstream based processes process the frame grabbed from the video stream and detect the position and orientation of all the nodes under each of them. The camstream processes then communicate the detected node position and orientation to a central process called the intermediate server. The camstream process only computes the position and orientation of the nodes within each camera. It is the intermediate server that translates this position update from each camera into a global position and orientation. It does this by keeping track of the fixed offset in the co-ordinate space associated with each camera. The current implementation of MiNT-m not take into account any camera's rotation. Cameras have to be aligned with the x-axis to have 0 slope. The intermediate

server is also responsible for averaging the positions of nodes that are located in the overlap region of two or more cameras to come out with a single value for its position and orientation. It is the source of node position/ orientation information for the rest of the MiNT system.

This separation of roles between the image processing and aggregation of results makes the system very scalable. The image processing algorithm in the camstream process is computationally intensive as each pixel has to be scanned. Only a limited number of camstream processes can be run on a given server. In the actual MiNT implementation, all 6 camera feeds at a resolution of 320x240 were processed by a single server. However, the testbed design should be scalable for arbitrarily large testbeds. To achieve this, wide angle cameras producing frames at a higher resolution or more number of cameras or a combination of the two could be used.

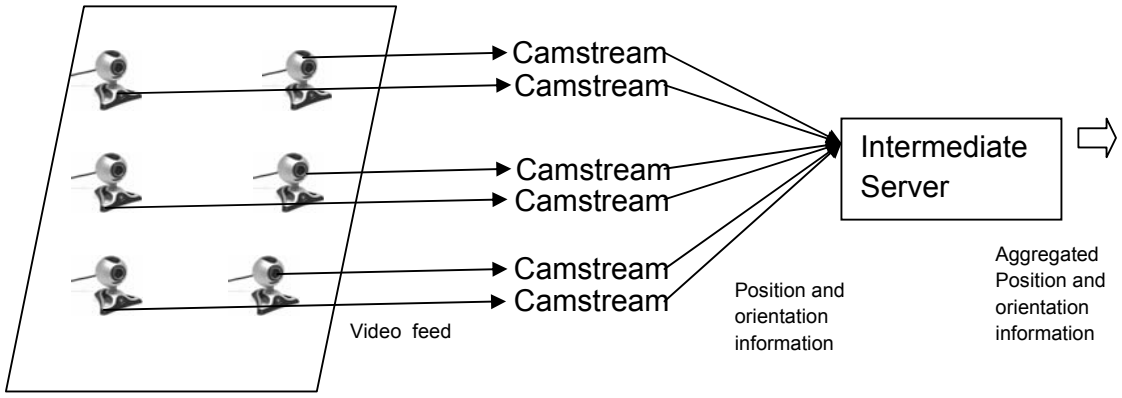

Figure 2 - Figure explaining the wireless architecture of MiNT-m wireless testbed.

In any case, the computational needs of image processing for a larger testbed would easily surpass the capabilities of a single server. Furthermore, there are hardware restrictions on the number of USB devices that can be connected to a single server. To overcome these difficulties, sets of cameras can be assigned to each physical machine. Each of them processes the video feed from a small set of cameras and sends the computed position and orientation to the tracking server.

1.1.2 The Vision Based Tracking Algorithm of MiNT-m

The vision based tracking system identifies the nodes' position and orientation using color patches on the nodes. The coloring scheme used by the node can be seen in *Figure 3*. To reduce the probability of false positives due to random objects in the test bed room being detected as nodes, the node incorporates a green header patch and red footer patch.

The header and footer apart from reducing the probability of false positives, also play a part in calculating the orientation of the nodes. The actual node identification is performed using the identification patches in the middle. A total of 4 colors dispersed as far from each other in the color space as possible are identified apart from red and green. The colors that were used for the original MiNT-m navigation system are 1.Yellow, 2.Pink, 3.Orange, 4.Blue yielding a total of 16 nodes.

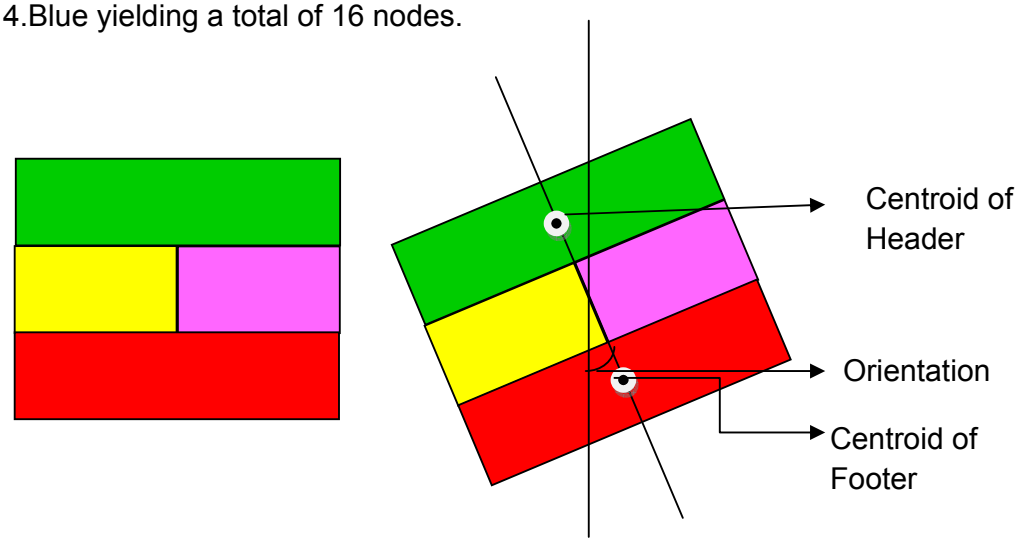

Figure 3 - Schematic diagram of the color patches on the MiNT-m nodes. The green and red colored patches at the top and bottom are the header and footer patches. The two colored patches in the middle are the ID patches.

The image processing algorithm used by the camstream based process is listed below. The following steps are applied to each video frame captured by the cameras.

The pixels in the image are normalized to lessen the impact of uniform lighting. Since uniform lighting scales each color by the same constant factor,

normalization works by dividing each color component by the sum of all components so that the constant scaling factor gets cancelled. The result can again be converted to the 0 – 255 scale by multiplying by 255 and rounding to the nearest integer. The normalization procedure follows the following formula;

$$r, g, b = \frac{r}{r+g+b}, \frac{g}{r+g+b}, \frac{b}{r+g+b}$$

Each pixel of the image frame is converted from RGB format to HSV format. The RGB color space is not used in analysis, particularly in color pattern matching applications because the values for Red, Green and Blue are correlated for most of the colors. As an example, yellow is a mix of red and green. HSV color space is used to obtain distortion invariant color image recognition. HSV ranges are pre-assigned to each of the colors in the color ID patches of the MiNT nodes. Note that although individual components of the HSV color space can overlap, there should be no two pair of colors that have overlap in all 3 color space components. This is to enable unique identification of a color.

Each pixel in the frame from the video is compared to each of the pre-assigned colors and if it matches any of them, it is classified as one of the colors of the color patch. Once identification is made, all neighboring pixels are scanned for a pixel that can be classified as the same color. Repeating this procedure for each new pixel identified results in small blobs classified into the same color. The remaining steps are executed after the image has been parsed once. Even if large blobs of the same color exist, they are typically identified by the camera as blobs of smaller size due to lighting irregularities and camera distortions. So, all blobs of the same color close to each other are coalesced into larger blobs. At the end of this step, blobs would have grown to the size of patches.

Once the patches are identified, only patches of a certain minimum size are considered. This dimension is larger for header, footer patches than for ID patches as they are larger in size. This is the first step in filtering out testbed noise. As a second phase of noise filtering, ID patches get associated with headers and footers closest to them. ID patches being smaller in size are more prone to noise. Hence, ID patches that cannot be associated with a header and footer near them are ignored.

Following this step, centroid is computed by averaging the location of all pixels for both ID patches and header, footer patches. The average of four centroids gives the position of the node within the camera. The slope of the line joining the header and footer gives the orientation of the node. The system is computationally efficient and simple to implement.

1.2 Thesis Proposal

We propose changes to the MiNT-m wireless testbed that includes a modified testbed design, flexible robot tracking algorithm and tools for easy setup of the MiNT-m testbed. The new design makes use of cheap, portable, off-the-shelf components to build the testbed thereby bringing down the costs and increasing the deployability. The new robot tracking algorithm was evolved with an eye towards ease of setup and effective use of the area under the overhead cameras. The algorithm uses a SIFT based image detection system that converts the images of the nodes on the testbed into feature points that can be used to identify the nodes uniquely.

This new scheme drastically reduces the setup time of the testbed and has a more relaxed requirement for camera alignment. The SIFT algorithm has an additional advantage of being more accurate and scalable to more number of nodes than the color based node detection scheme. We also developed tools for automatic image selection for the SIFT module and a tool for automatically detecting the position of cameras in the testbed and help in setting up the cameras to cover the entire testbed. The objective of this thesis is to reduce the testbed setup time significantly to the extent that the testbed can be setup on demand within a couple of hours, costly.

The rest of this thesis report is organized as follows. In Chapter 2, we discuss the node localization algorithms used by contemporary wireless testbeds and setup process involved in setting up each of the testbeds. We also discuss the problems in the node localization algorithm and a few related setup problems of the MiNT-m testbed. In Chapter 3, we discuss the proposed solutions to the problems discussed above. In Chapter 4, we discuss different performance metrics for the node localization, camera positioning system (a tool for easy setup of MiNT-m setup), and the time taken for setup. In Chapter 5, we conclude the thesis and discuss future improvements to some of the proposed algorithms.

Chapter 2

Related Work

Several wireless testbeds had been developed and deployed throughout the world. However, only a handful of them address the problem of mobility and the related problems of tracking and setup. Some of the more famous deployed wireless testbeds and how they tackle the problem of movement and tracking has been discussed.

2.1 Testbeds with Virtual Movement

Testbed setups like ORBIT [1] use a fixed array of wireless nodes for their experiments. In case of ORBIT, the testbed consists of 20 X 20 nodes for a grid of 400 nodes. The movement is simulated by migrating the virtual state of the virtual node used for experiment from one node to the other across the nodes in the grid lying on the proposed path. Movement achieved in this case is at discrete intervals. Also costs and time involved in the setup of a similar testbed are very high.

2.2 Testbeds with Optical Tracking

Several testbeds use variations of color based tracking or color based tracking in combination with other heuristics to determine the position and orientation of the wireless node.

Mobile Emulab [2] uses Acroname Garcia robots mounted with XScale based Stargate small computer running Linux with an attached Mica2 Mote as the mobile wireless testbed. The testbed uses an array of ceiling mounted cameras that cover the entire testbed. The modified Mezzaine vision algorithm then recognizes a pair of differently colored circular blobs attached to each of the robots. The blobs are of the same two colors (colors widely separated in the colorspace) for all the nodes. So the vision based tracking system is only used to identify the location of the nodes in the testbed, not to map each node to its location. This mapping is achieved by the “Wiggle

Algorithm", moving nodes one at a time to map a particular node to their position. This procedure is executed during the testbed setup at the start of the experiment. Thus the testbed setup time increases linearly with the number of nodes in the testbed as only one node can be moved at a time. Also, the visiond tracking system aggregates the feed from multiple cameras into a single track to handle the case when nodes are located at the boundary of two or more cameras. This step also introduces scalability concerns. Also setting up the overhead cameras also involves significant effort without the use of additional tools to aid the setup.

The Caltech Multi-Vehicle Wireless Testbed [3] uses stripped down laptops on custom built robots powered by ducted fans. The testbed is setup in a special room with no black spots. Black blob patterns on each of the nodes are used to uniquely identify the nodes. Also the angle between certain sized blobs are used to determine the orientation of the vehicles. This setup requires a specialized room for the testbed. Also, the number of combinations of uniquely identifiable patterns that can be formed within the area occupied by a node is limited, limiting scalability. This is because, in addition to providing uniquely shaped/positioned blobs for unique identification, certain portion of the image has to be dedicated for redundancy/validation to prevent confusion when nodes move close to each other.

Kansei: a high-fidelity sensing testbed [5], primarily a testbed for sensor networks uses a array of static nodes as well as mobile nodes for experiments as well as for injecting sensor readings. The setup consists of an array of stationary nodes supporting a Plexiglas platform on which the mobile nodes are run. Each mobile node is consists of Acroname robots with 802.11b and XSM nodes. The layered setup is required to support their unique requirement of having to inject sensing events in real-time. The mobile nodes apart from running experiments are capable of producing light that passes through the Plexiglas to activate the sensors of the static nodes that are situated below the Plexiglas. To enable this, the mobile nodes are tracked using a high resolution, network programmable cameras with capabilities for pan, tilt and zoom to focus a prescribed location on the testbed. The video feed from this camera is used by the vision system to track the mobile robots.

However, the system may not be scalable in a testbed with multiple nodes moving simultaneously.

2.3 Testbeds with Non-Optical Tracking Systems

MiNT-2, A miniaturized robotic platform for wireless protocol development and emulation [4] is an improvement over MiNT-m wireless testbed. The optical tracking system is replaced by a three tier tracking system. The first tier makes use of the iRobot create robot's ability to move at a particular speed and orientation. The second level uses the robot's rotation sensor that can be used to precisely measure the distance moved as well as the rotation of the robot. However, the tracking system cannot be designed purely based on the first two tiers. Both the systems can lead to accumulation of error in both position and orientation. The errors could be caused by slippage in the robot's wheels, rounding errors or due to encoder inaccuracy. The third tier is composed of an array of RFID tags distributed throughout the testbed. The position of all the RFID tags are already known and when the robot enters the reading range of any of the RFID tags, the position of the robot can be determined. Orientation of the robot is calculated by the robot itself every time it passes through two RFID tags. The angle of the line joining the two RFID tags is added to the rotation reading from the rotation sensors in the distance travelled between the two RFID tags. The setup for this testbed involves placing RFIDs on the floor, configuring their position and setting the robots to auto configure. In the auto configuration mode, the robots travel in a straight line, till they come across two RFID tags, thereby establishing their position and orientation. During the auto configuration phase, if the robot comes across an obstacle like a wall or another robot, it takes a random rotation and continues its straight line path. The auto-configuration phase is considered complete only when the node manages to hit two RFID patches while moving in a straight line.

Though the array of RFID tags is cheaper than a sparse array of cameras, this configuration has a few disadvantages. During the auto configuration phase, since the position of none of the nodes are known, there

is a high possibility of collision among nodes, if the density of nodes in the testbed is high. Also, since the probability of collision increases with the increase in the number of nodes, the testbed setup time also increases with the increase in number of nodes in the testbed. Another minor problem is that a small portion of the battery charge is used up during the auto configuration phase.

Chapter 3

A New Computer Vision-based Robot Tracking System

3.1 Motivation for the New Optical System

We tried to retain a vision based tracking system as it has some inherent advantages over other de-centralized tracking systems. The vision based tracking system can optionally allow the remote user to view the testbed to actually verify the actual position of the nodes. Also, a central optical tracking system would still be able to track the position of a node even if the router board on the node has crashed (A common occurrence in a testbed with large number of nodes and continuous operation) and there is no communication from that node. This would enable the current dead node to be classified as an obstacle so that other nodes can navigate the testbed without colliding with this node.

The primary problems we had been trying to solve are threefold.

1. Replace the color based identification algorithm with a more robust and scalable vision based tracking system.
2. Eliminate or reduce the requirements for precise position of camera.
3. Try to reduce by as much as possible the area wasted by the overlap requirement.

3.2 Options for Improving the Color-based Node Positioning Algorithm

The initial idea was not to rely on colors but on image features for object identification and detection. We considered several alternatives with this objective. Using Alphabets and or numbers printed on the patches, using 2 dimensional bar codes like QR codes or using distinct images on each node and using an image detection algorithm.

The first idea that was considered was using alphabets and or numbers and using OCR like algorithms to detect the image. Thought the algorithm to detect images or numbers is pretty straight forward, this scheme would only

work if alphabets/numbers are fed to the algorithm upright and individually. Even if this were possible, we would only be able to identify that a particular node is present in the image that was input to the algorithm. A different algorithm then needs to be used to determine the exact center of the node. As for orientation, it has to be determined even before the frame is fed to this algorithm as the image being fed to this algorithm should be adjusted for orientation and made upright.

The second idea was to use 2 Dimensional bar code encoding formats like QR codes. An example of QR code can be seen in *Figure 4* [6]. Tools for generating thousands of unique bar codes are already available. The algorithm is fairly resistant to changes in scale and rotation. However, an independent scheme has to be used to detect the orientation of the image as well as the boundary of the image to compute the center of the node. This is required because the algorithm requires a rectangular image from which it will try to read the store information. So, only if the node's boundary is known, the image within the boundary can be input to the QR code decoder.

Figure 4 - Example of a 2D binary code, QR code [6]

The other option was to use a set of images on the nodes and use an image detection algorithm to classify the images into the set of already decided training images. This way, we could have an infinite supply of images.

For this procedure, we considered the SIFT, Scale Invariant Feature Transform algorithm. The algorithm has very good resilience for changes in scale, rotation, lighting and affine transformations. This means that the results produced by this algorithm are very accurate as well as resilient to errors.

Also, since the algorithm uses the image features to detect the images, the difference in orientation between these image features can be used to determine the orientation. Also since a number of feature points are used to obtain orientation, and the feature points themselves match only if the feature is found in the image, the orientation measurements from this algorithm is also of higher accuracy. Lastly, an image is classified as a particular image only if certain number of features in the training images are present in the image captured by the camera. This guarantees that the probability of false positives or incorrect detection is very low. The only problem with this algorithm is that it is computationally more expensive compared to the other methods.

Given the requirements of the testbed to be resilient to lighting changes, scale and orientation SIFT was chosen as the algorithm for the optical detection system. We used a implementation of Sift from Rob Hess of Oregon State University [8] in our optical node system.

3.3 Scale-Invariant Feature Transformation

Sift algorithm is mainly used in computer vision for identifying objects in the real world that were already present in a set of training images. The actual steps of the algorithm are listed below [7].

The first step of the algorithm is to find scale-space extrema. A series of images are constructed in different scale-spaces and Gaussian transform is applied to them. After this step, Laplace of Gaussian gives the best indication of the characteristic scale. However, to reduce computational complexity, difference of Gaussian is performed instead. After this step, each pixel is compared with each neighboring pixels including the pixels in the neighboring scale to obtain the Difference of Gaussian Extrema.

This step involves locating the true extrema by taking the derivative of the Taylor's series. Also filter out points with low contrast and perform edge response elimination. The next step performs the orientation assignment. For each region around the key points identified in the previous step, create a

gradient histogram with 36 bins for orientation weighted by Gaussian window. All peaks above 80% of the highest peak are used for creating key-points with that orientation.

Using the image of the closest scale, rotate a region of pixels typically 16X16 pixels close to the computed key points, by the already determined orientation. This region is divided into sub-regions typically, 4X4 pixels for a total of 16 sub-regions. In each sub-region, a gradient histogram is drawn with 8 bins for orientation. This 4X4 sub-regions each with 8 bins forms a 128 element vector referred to as feature point.

Feature points are computed individually for each of the images used for comparison. All comparisons or matching operations performed on the image are essentially matching the feature points. Typically, the feature points from the training image are stored in a K-dimensional indexing structure like the K-d tree to make the comparison process faster. The algorithm is made scale invariant by considering multiple scale-spaces, rotation invariant since only the best orientation is considered.

3.4 SIFT based Robot Tracking System

The SIFT algorithm as explained above can be used to match objects in training images to objects in the frames from the camera. However, this information should be used to obtain the exact position and orientation of objects. Though well established algorithms are already available for outlier elimination, we use a computationally simple outlier elimination algorithm that delivers sufficient accuracy of operation.

3.4.1 Robot Tracking Algorithm

The devised algorithm is explained below:

A set of training images are selected without having any common features. Given any two random images, there is only a small probability that the two images will share a significant number of features. We also wrote a tool to make the process of finding images for the SIFT algorithm. The tool is

discussed in section 3.4.2. The training images are pre-processed by the training algorithm to identify all the features in them. Each image is associated with a set of features each of which includes the position of the feature, the orientation of the feature as well as SIFT related information required for matching these features with the features on the destination image.

Each frame from each individual camera is broken down to feature points. These feature points are then compared to each set of features in the training image. The matching features are maintained in a separate set for each training image. It has to be noted that multiple MiNT nodes can be present in the same camera frame. This is the reason for maintaining separate matching set for each image.

However, in this set we need to maintain only the position and orientation of each feature. We will not be using SIFT based attributed after this point in the algorithm. For further computation, we also associate each feature (position and orientation) in the matching set with the feature point in the training image that matched it. Here also, it is sufficient to maintain only position and orientation of the matching features in the training set. Let us refer these training feature points associated with features in the matching set as associated training feature points.

Once the feature points are identified, centroids for the MiNT nodes are calculated from them. This step is necessary as features tend to be concentrated along the rich portions of the image. Taking an average of the position of the features would skew the value of the centroid towards these rich regions.

The computation of the centroid for the MiNT nodes requires the use of the associated training feature points. Thus the output of this step is a set of centroids as computed from each element of the matching set. Let us call this set the centroid set. At the end of this step, there is utmost one set of centroid feature points for each training image. In other words, nodes that are present in that frame would have a centroid feature set associated with it. Ideally, Nodes that are not present should not have centroid sets associated with them. The computations for calculating centroid of the MiNT nodes are

discussed in section 3.4.3. After the completion of this step, the associated training feature points need not be maintained.

However, there is a small probability that the nodes that are not present in the current frame could have centroid sets with a few points. It means that few features in these images look similar to features in other training images. On the same note, there is also a small possibility of a some features in the matching set of correct nodes actually being features of other images or objects in the testbed. In both of the above cases, the number of erring features are typically small, one or two.

However, taking them into account for further calculation would lead to incorrect values for the centroids. These points can easily be eliminated as they typically are well separated from the correct set of points. Also, these points could have incorrect orientation which could skew the orientation significantly if the actual computed orientation happens to be a small value. The algorithm for elimination of these points is simple and light-weight. (Discussed in section 3.4.4)

After the elimination of erring points, if the number of feature points in the matching set (when compared with a particular training image) are higher than a particular threshold, we assume that the feature points in that matching indicate the presence of that particular node at that position.

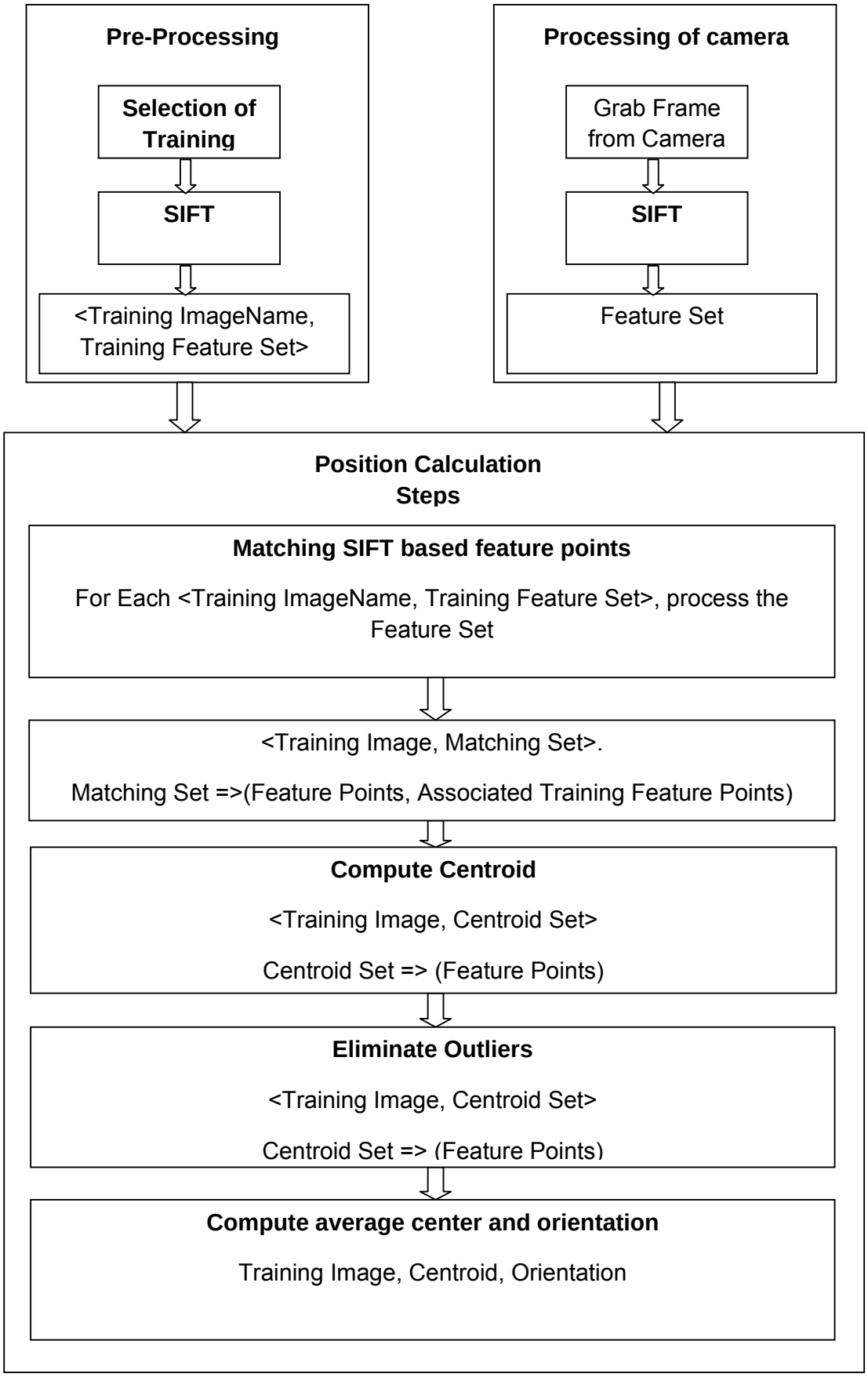

Figure 5 - Flowchart of Node Localization algorithm.

Now the center of the node is given by the average co-ordinates of all the remaining features in the centroid set. The orientation is also computed as the simple average of all the orientations. However, some special cases are needed to handled for orientation. This occurs when angle is close to 0. The average orientation is computed as 180 as the average between 0 and 360. As of now, this issue is fixed by detecting this case and handling it specially.

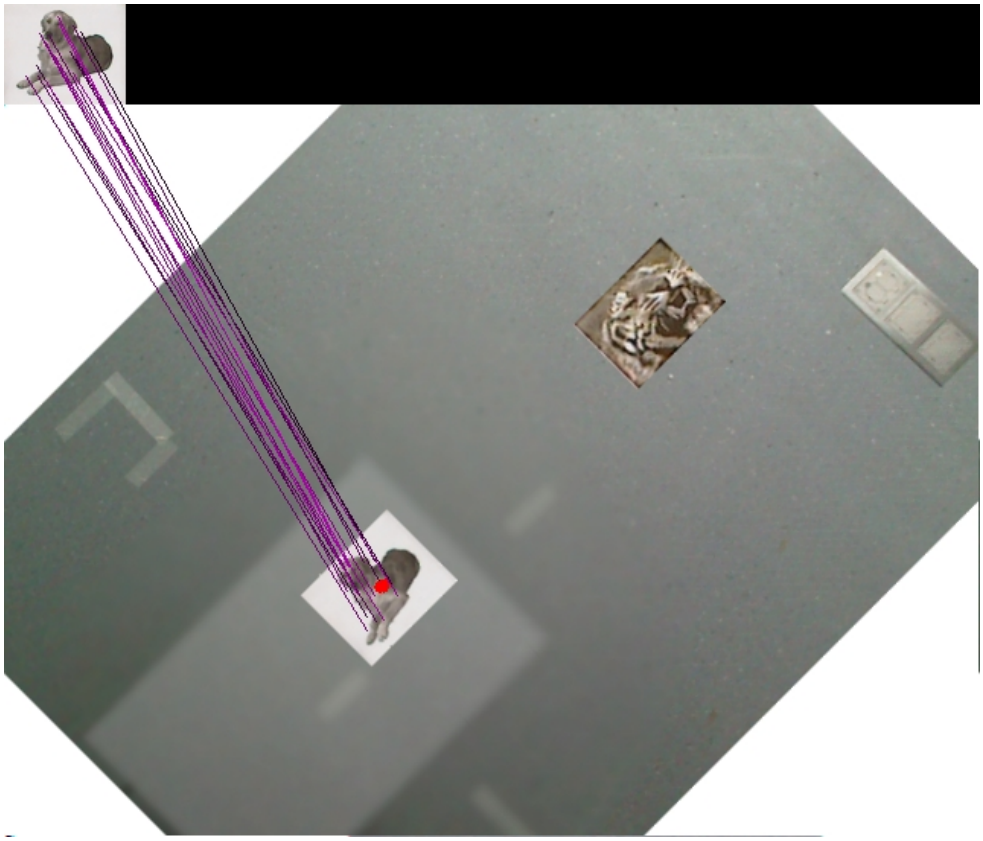

Figure 6 - Training image dog matching a dog in the testbed. The red dot illustrates the computed center.

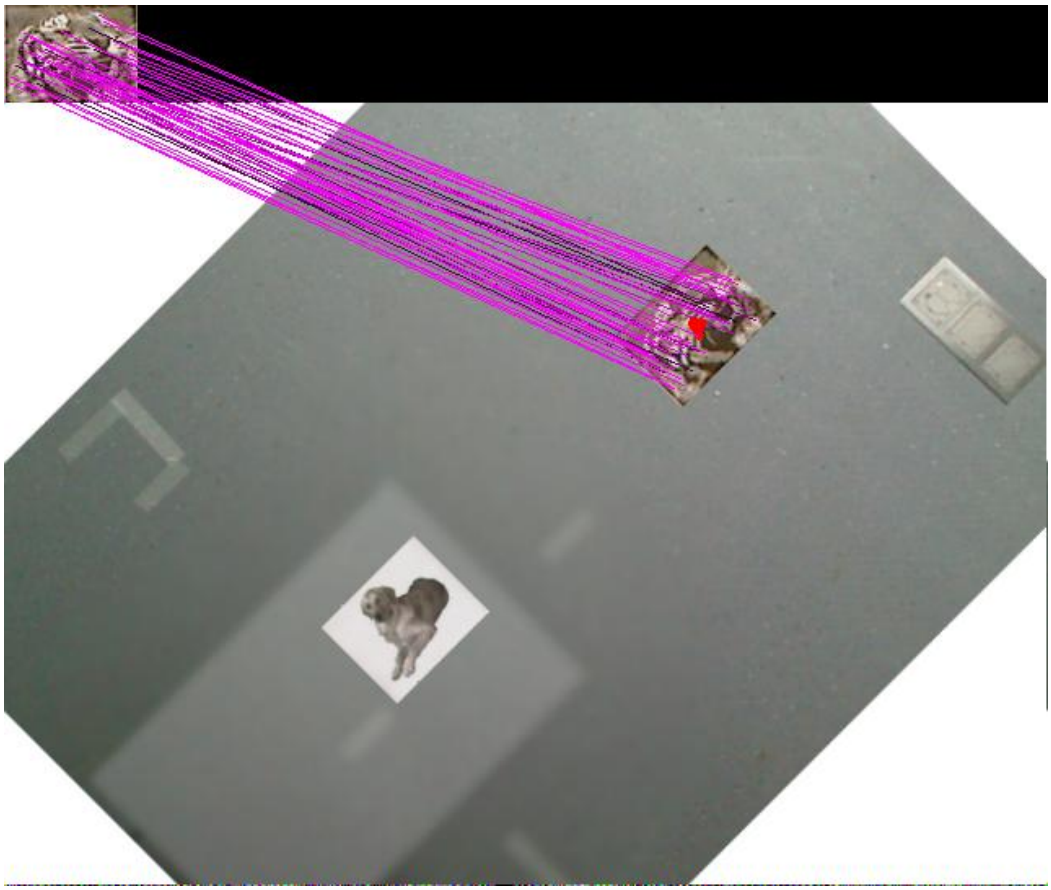

Figure 7 - Training image tiger matching a tiger in the testbed. The red dot illustrates the computed center.

3.4.2 Tool for Automating the Selection of Training Images

Though the process of selecting training images is easy compared to other vision based recognition systems, it can become difficult when the number of images are high. Also, if the entire process is manual, it is possible to choose an image which has features similar to several different images if there are large number of images.

So, we evolved an algorithm that when given a large folder of images, could automatically process them using the SIFT algorithm and select the best set of images that could be used. There are 3 parameters that determine the desirability of a image for the SIFT algorithm 1) The image must be composed of a minimum number of feature points. 2) The image must overlap with as few other images as possible. 3) The overlap with any other image in

the set, in terms of number of feature points should be kept below a threshold. 4) Image should not have repeating features within itself. Though, this does not impact the image detection in anyway if all the above criteria are satisfied, there is a possibility of features within the images being matched to other features, thus making the orientation computation inaccurate.

The selection algorithm consists of 2 stages. The first stage is the filtering stage. Each image is fed to the SIFT algorithm to generate a set of SIFT feature points. Images having number of SIFT feature points below a certain threshold are skipped in this step. Also, images are matched to themselves and their center and orientation computed. If their center and orientation are significantly different from their determined orientation, they are eliminated. This step is to filter out images with property 4.

The second step is responsible for enforcing the steps 2 and 3 of the desirability parameters. First, the SIFT feature points of each image is compared with the feature points every other image and the number of feature points overlapping with every other image is determined. We store the number of feature points from each image overlapping with other images. We also store the total number of overlapping feature points.

It is to be noted that the acceptable threshold is calculated individually for each image depending upon the total number of feature points it consists of. In other words, nodes with large number of feature points can afford to have more overlap, than nodes with lesser number of feature points. The reasoning is that, in case of nodes with larger number of feature points, after the overlapping points get eliminated by the outlier elimination algorithm, these nodes would still have enough feature points to qualify as a valid image. (The minimum number of feature points required in the optical tracking algorithm to qualify as a node is constant.)

> *Acceptable overlap = Number of feature points – Minimum number of feature points needed to qualify as a node.*

Technically, an image overlapping in features with many images just above the acceptable threshold is far worse than an image which overlaps

with a fewer number of images with a larger number of feature points. The reasoning behind the above statement is that the former image disqualifies a larger set of images from being usable. However, such nodes tend to occur less frequently. Such nodes should be eliminated first.

To detect this case, during the calculation of step 3, a count of the number of images disqualified as a result of this image is also maintained. Eliminate the image that disqualifies the largest set of images and adjust the values for the rest of the images. In case of more than 1 node with the same count (occurs frequently when the count drops to 3 or below), the conflict is resolved by using the number of feature points overlapping as a percentage of the total number of feature points in the image.

The above steps are executed till a) No images are being disqualified. b) Number of non-overlapping feature points for all images are greater than the threshold. Once this is complete, the images can be used for the optical tracking system. At each step when a node is eliminated, the stored values for each of the nodes is to be adjusted. This adjustment step takes only linear time. In a single pass through the cells of the offending node, the number of feature points overlapping with each node is subtracted from the total. Also, subtract 1 from the count of number of images disqualified when this node is removed.

3.4.3 Computation for Calculating the Centroid of the Images given the Matching Features

This computation requires both the location, (position and orientation) of the current feature point as well as the location of the feature it matches to in the training image. In addition to this, it requires the position of the centroid of the training image. Since training image only contains the node image, the centroid is always image width/2 , image height/2. Since the distance and angle of the centroid of training image to the matching feature in the training

image is already known, we should be able to compute a similar point for the actual image offset by an angle given be the orientation of the image.

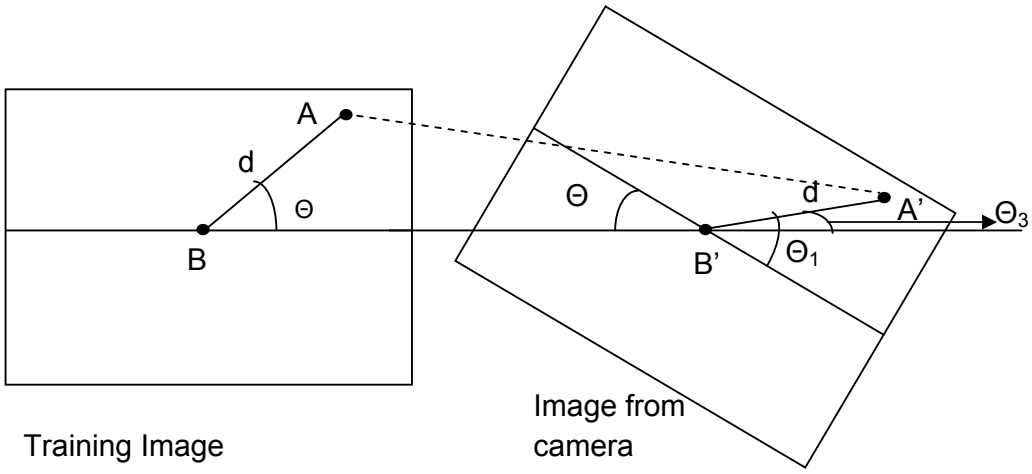

Figure 8 - Schematic diagram of computation of center B from each feature point, one of which is shown A.

Using points A and B, the distance (d) between centroid of the training image (B) and the mapped feature point on the training image (A) is determined. Also the slope of the line joining A and B is determined. (Θ). However, this slope has to be adjusted since the origin of the current testbed is in the top left corner. Comparing the images of the two nodes in *Figure 8*, the slope Θ is retained as Θ_1 even in the rotated image. In addition to that, there is an additional component of rotation caused by the orientation of the orientation of the node. Therefore, the center on the image from the camera is computed by:

$$Offset\ factor\ (X,Y) = d * \cos(\Theta_1 - \Theta_2)$$

The offset factors can then be added or subtracted to the co-ordinate at point B' depending up on the quadrant of occurrence of B keeping A as the origin.

3.4.4 Computation for eliminating Outliers

The algorithm takes advantage of the fact that outliers are generally distributed away from the centroid of the image and tries to eliminate such points. However, there is a small probability that the feature matched a wrong point, but still lies close to the centroid. Such points do not affect the average position of the centroid by a large factor. So we do not take steps to eliminate these points.

For each feature in the image set, compute the Euclidean distance with all other points in the matched set. Compute the median of the distances. If the median distance is greater than a particular threshold (typically half the size of the node), the point is not considered for node identification. The reasoning is that if it is a genuine point, it should be located close to the actual centroid. So, this distance with most of the other points must be small and median would be one of those values. This results in the genuine point being selected.

However, had it been an outlier, it would be well separated from the centroid, and therefore will be far from most of the other points. Even if the current point occurs with a small group of outlying points, the median would be one of the distances between the current point and one of the points close to the centroid which would be large. As a result, this point would be eliminated by the outlier elimination algorithm. Median is used at this stage instead of mean because values are typically small and a few rogue values can skew the calculated values significantly.

Chapter 4

Automatic Camera Array Calibration

4.1 Problems in Calibrating the MiNT-m Color Based Tracking System

Nodes in the MiNT tracking system are identified by the color of the ID patches attached to them. For this scheme to work perfectly, all the colors used for the patches in the test bed have to be as far apart in the HSV color space as possible. If the colors chosen for the ID patches are widely separated and as non-overlapping as possible in each of the components H, S and V, each of the colors can have a wide range of values that would still match to this color. This would take care of minor changes in lighting conditions, camera blurs at the edges or the effect of shadows.

However, only a limited number of such unique colors can be identified as the colorspace is constant. The MiNT-m testbed uses only 6 colors of which two colors - green and red are used for header and footer. The remaining 4 colors each of which can be used in any of the ID tags giving rise to a total of 16 nodes. Even with such a limited number of colors, the probability of a color being identified incorrectly or not being identified at all was a small but significant value. So the only way the testbed could be scaled is to increase the number of patches used for identification. This would have to be accomplished keeping the area of each patch constant, leading to an increase in the total area of the patch itself. This would in-turn lead to reduce the total area available for experiments both directly as well as indirectly (requirements of overlap and collision avoidance.). In effect, the color based node identification scheme does not lend itself to scale for hundreds of nodes.

The color based identification scheme is also very susceptible to variations in lighting and shadows. The scheme is so specific to lighting in a particular area, that each camera in the same testbed has different ranges of HSV values for each color owing to the slight difference in lighting across the testbed. This in-turn leads to incorrect detection of nodes at the borders of the

camera and also at places where there are changes in lighting within the same camera. Coming up with range values for each of the cameras is an imperfect and non-trivial task. Typically, all the color patches are placed in a distributed fashion throughout the testbed under a camera (Each of the cameras have to setup individually to handle variations in lighting). This is followed by coming up with different values for the ranges till all the nodes are identified.

We wrote a few tools to help make this process easier. The tool has two components to it. The first component takes as input the different positions of all the colors. As an example, if the current camera has six color patches placed under it, green would be present in six different locations as header, red in 6 different locations as footer and the different ID colors in different location on the ID patches. With this input, it generates statistics on all the colors including mean, standard deviation, median, max and min values for the H, S and V components of all the colors. Looking at this statistics for each of the components, first, an attempt is made to assign HSV values to all colors without overlapping using the min and max values. In most cases, this is not directly possible. In this case, mean and standard deviation values can be used to come up with slightly compromised range values for the HSV value components.

The second part of the tool allows us to see how the camera has recognized the color patches for each of the range values being tried. The two part tool makes the job of coming up with guesses for range values slightly more data oriented. However, the process was still cumbersome and required significant amounts of time to tune the HSV color ranges to the optimal values.

The MiNT testbed requires the cameras to be aligned in a particular way to work flawlessly. The cameras should be aligned to either of the axes and or the border of the other cameras as closely as possible. Also, the overlap between cameras at the intersection should be atleast as large as one MiNT node. Both these tasks are non-trivial and require considerable manual effort. This effort becomes even more difficult as the height of mounting the

camera increases. The difficulty also increases as the testbed becomes squarer, as overlap requirements have to be satisfied with the four adjacent cameras.

The overlap requirement also wastes a lot of usable camera area though overlap allows us to avoid stitching thereby increasing scalability. The amount of camera area wasted by the overlap can be computed by the formula

*roomba_diagonal * (rows - 1) * camera_height + roomba_diagonal * (columns - 1) * row_width*

where *rows, columns* represent the number of rows and columns of cameras,

camera_width and *camera_height* represent the camera resolution

In addition to this, for each column or row of cameras, the node cannot be allowed to move out of the test bed partially. This would render another

2 camera_width * roomba_diagonal* unusable for each column and *2 * camera_height * roomba_diagonal* unusable for each row. So, the total camera area wasted by the testbed is approximately given by:

*= roomba_diagonal * (rows+1) * camera_height + roomba_diagonal * (columns+1) * row_height*

*= roomba_height * [(rows + 1) * camera_height + (columns + 1) * row_height].*

However, it is to be noted that all test bed would waste an area equal to the second factor to maintain accuracy in node position at the testbed borders.

4.2 Goals for the New Calibration System

As previously discussed, the problem of perfectly aligning the camera is two-fold. Any optical tracking system for a testbed has to cover the testbed in its entirety. This requirement cannot be avoided. However, achieving this task without wasting too much camera area is a non-trivial task. We developed a tool for simplifying camera based setup to ensure full testbed coverage.

Adjusting the camera for the above MiNT-m setup also involves the more difficult task of satisfying the requirement of overlap. Ensuring exact overlap makes the setup difficult and time consuming apart from wasting effective camera area. Ideally, the tracking algorithm should work without the need for overlap. This would not only reduce the setup time drastically, but also help reduce the amount of camera area wasted by camera overlap.

4.3 Design of the Automatic Camera Calibration System

4.3.1 Ensuring full testbed coverage

To ensure, full testbed coverage, the calibration system has to know the dimension of the testbed and compute the precise location and orientation of the cameras in the testbed. If this information could be obtained, we could then determine the exact location of camera within the testbed. Once we have this information, a unified image of the testbed showing the orientation of position of all cameras in the testbed can be displayed for the user. Also, the displayed image can be updated whenever the user moves a camera manually. The visual representation of the testbed as well as online feedback showing the current status of the testbed in response to him moving the cameras will make it easy for him to setup the cameras. Uncovered regions of the testbed can be easily identified and corrective action taken.

Handling the Camera Area Outside the Testbed

It must be noted that cameras at the borders of the testbed with incorrect orientations can have portions of the camera area present outside the testbed. This portions of the camera area should also be displayed to make it is easier for the user tuning the position of the cameras. Ideally, the user would want to minimize the amount of camera area being wasted. Displaying the camera area outside the testbed allows the user to adjust the camera so that this area can be minimized. To make this possible, the testbed image being referred to in the automatic calibration system uses a constant percentage of the total dimension of the testbed as a border. In our

experiments, we used a border of 30% which effectively means that there is a 50% left, right, top and bottom margin.

Automatic Camera Calibration

When the user is done with the calibration process, the position and orientation of all cameras in the testbed is written to a configuration file. This information is then used by the tracking algorithm during the tracking process to determine the location of all the cameras in the testbed. The images / features detected under each camera are adjusted with offset and orientation of the current camera. As a result, this tool along with the improvements in the tracking system completely eliminates the need for exact camera alignment.

Eliminating the Need for Overlap

As previously discussed, overlap is required only to ensure that the nodes are fully covered by at least one camera. However, using the properties of the SIFT algorithm and clever design of tracking components, the overlap requirement can be completely removed.

SIFT detects images by matching features generated by the training image with features in the current image. Once a match is established, the position of the node's center and its orientation can be determined with respect to that feature. This computation can be performed even if this feature is the only feature detected by the current camera. Given this property, all features detected by each camera are sent to a central system, tracking server. The tracking server then groups the feature points based on what training images they originally matched and then computes the center and orientation. This procedure completely eliminates the need for overlap area between cameras.

As a result, automatic calibration system can direct the user to just fully cover the testbed without the need for worrying about a minimum amount of overlap. Also, the elimination of the overlap and automatic computation of camera positions on the testbed means that the cameras can be positioned any way fit such as to minimize the amount of overlap and wastage of camera area.

4.3.2 Design of the Automatic Calibration Process

Different approaches were considered for determining the position of the cameras. The approaches included numbering the testbed floor, drawing a grid on the testbed floor etc. However, since we already have an image recognition system in place, we decided to leverage the capabilities of the SIFT algorithm to locate the camera on the testbed. Images are placed in the testbed at regular intervals in known positions which is used by the algorithm for its computations.

It is to be noted that the calibration process should detect the pixel to inch ratio, the position and orientation of camera on the testbed. For the detection to be accurate, each camera must have at least two images lying in its range of vision.

Algorithm for Automatic Calibration System

The algorithm takes as input the frames from multiple cameras, the dimensions of the testbed in inches and the position of all the images in inches. It also takes as input the size of the canvas to display the testbed and location of the cameras. This size is typically the maximum size of the image that can be displayed in a window (Browser). It processes each frame of the camera individually and uses the position and orientation information to update a global image of the testbed. The global image of the testbed is reset at the end of each run.

The testbed is represented by a rectangle of a different color (white in our testbed) and the rectangle corresponding to the cameras (blue with red borders in our case) are drawn over this rectangle. At the beginning of each cycle, the rectangle corresponding to the testbed is drawn on a clear image and the following processing steps are performed for each frame of every camera:

The image is fed to the SIFT module where it is compared with every training image in the set. The SIFT module returns the name of the images that are currently present along with their current position (in pixels) and orientation within that particular camera frame. The first task of the automatic calibration system is to identify the pixel to inch ratio for each camera. There are differences in pixel to inch ratio for each camera owing different mounting heights and or different levels of zoom. Since the task of camera position locator requires processing the location of different cameras and representing them in a single image of fixed resolution, most of the calculations are preformed in inches. This is because different cameras have different values of pixel-to-inch ratio and the pixel values from them cannot be compared with each other. At the final step, the inches are mapped to fixed dimensions of the testbed image.

To find the pixel-to-inch ratio, distance in pixels is computed between the centroids of each pair of images lying under the camera. Since the absolute position of all the images in the testbed is already known, distance in inches can be computed between each of these pairs can also be readily computed. The pixel to inch ratio is obtained by dividing the distance in pixels by distance in inches for each pair of images and taking the average.

$$\text{Pixel-to-inch ratio} = \left(\sum_{i,j} \left(\text{Pixel distance between centroid}_i \text{ and centroid}_j \;/\; \text{Inch distance between centroid}_i \text{ and centroid}_j\right)\right) / \text{Number of pairs of centroids.}$$

The next step is to compute the position of the center and orientation of each camera. For this step all positions are converted to inches. This procedure is very similar to the computation of center for the images in the node location algorithm. Let us first look at the scenario within the camera.(Please refer *Figure 9*). The center of the camera can be readily computed as the frame resolution width/2 , frame resolution height/2. Let the distance of the centroid of an image to the center of the camera be represented d. Similarly, the angle of the line joining the centroid to the centroid of the camera can also be readily computed, say Θ. Now if the

calculations have been computed in inches so far, the distance between the camera center and the image centroid will still be the same. (d). Moreover, the position of point A' which represents the absolute location of the image in the testbed is already known. From this information B', the center of the camera has to be computed. This is the same problem as finding the center of an image on the testbed, given the position of the point and the center in the training image. Please refer section 3.4.3 for the calculations to determine the position of B'.

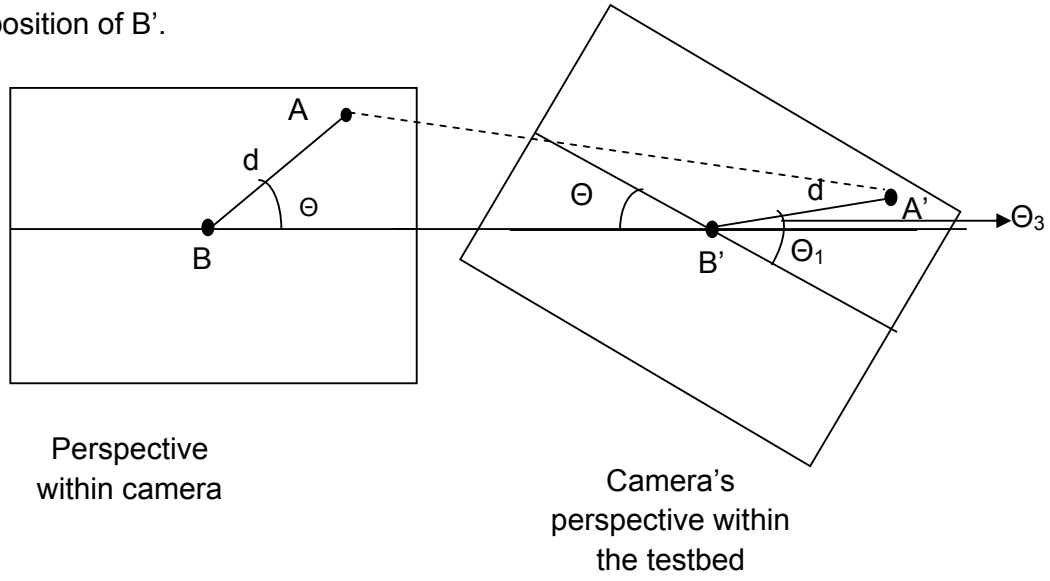

Figure 9 - Illustrates the calculations for computing the center of the camera in the testbed.

Following step 5, center is computed with respect to each image present in the testbed. This value is then averaged to determine the center of the camera. Calculating orientation or the angle of tilt for the camera is trivial and can be computed by averaging the orientation values for each of the images.

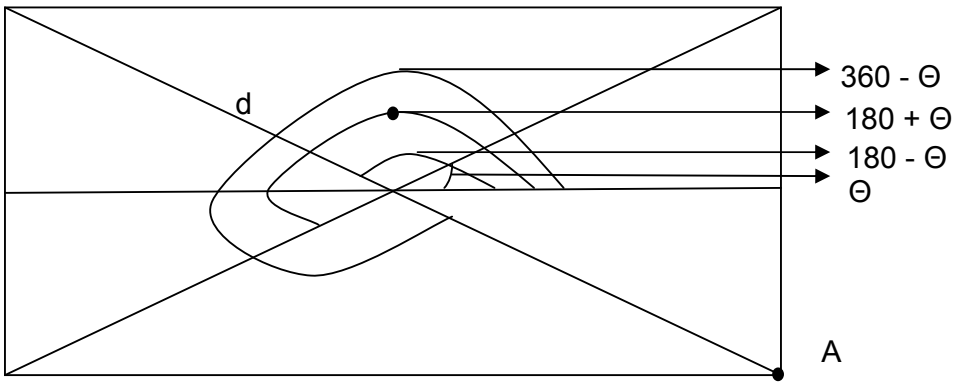

Figure 10 - Schematic diagram showing the computation of the four corners of the rectangle.

The next step of the algorithm computes the four corners of the rectangle using the center and orientation just computed. Using the pixel-to-inch ratio and the resolution for the camera, the bounds of the camera in inches can be computed. The four corner points of a rectangle can easily be computed given the center point, its orientation and the length of the half-diagonal (d). Like in the previous step, two offset factors are computed for each corner based on the angle of that corner from x-axis and the angle of rotation of the camera. For example to compute the position of the point A,

*Offset_x = d * cos (360 – Θ + orientation of camera)*

*Offset_y = d * sin (360 – Θ + orientation of camera)*

A (x,y) = (camera center X + Offset_x, camera center Y + Offset_y)

Once the four corners of the rectangle are obtained, lines are drawn joining the four corners of the rectangle.

4.3.3 Hardware Components of the New MiNT-m Testbed

The new MiNT-m testbed was designed to be setup using as less time as possible and also be highly reconfigurable. So instead of attaching the cameras to the ceiling, we decided to attach them to rods (extension poles) supported on each side by props. For props, we extended the height of commercially available coat-stands to increase their height to 9 feet. We then placed an extension pole on top of them. Cameras are then attached to the extension pole. To prevent the coat stands from toppling over as they have become top-heavy, weights are attached to them. This is the procedure for constructing a single row of cameras. This can be extended to any number of cameras by repeating the same number of steps.

4.3.4 Software Components of the MiNT-m Node Localization Component

The MiNT testbed is composed of composed of primarily of 3 main software components

1. Camera server
2. Camera Position Locator
3. Tracking server

Architecture of the Camera Server

The camera server is responsible for computing the feature points lying under each camera. One camera server process is run for each camera in the testbed. An open source tool named streamer is modified to capture video frames from the camera. The camera server process waits on the socket for being polled. On being given the region of the image under the camera to scan and the training image to compare with, the camera server captures the current frame from streamer, performs the SIFT based matching of the portion of the image specified, with the features in the specified training image. This results in features matching the training image. Using these features, and the computation detailed in section 3.4.3, the center of the nodes according to

each of the feature points can be computed. These feature points are then returned to the polling process.

The polling process can be either the camera position locator or the tracking server. The functionality of the camera server remains the same. The camera server is designed as one process for each camera to increase scalability. The bottle-neck in both node localization and camera position locator is the SIFT algorithm for matching features. If the process of grabbing the entire frame and processing it for SIFT can be moved to different physical machines for different sets of cameras, the setup can be scaled to tens to hundreds of cameras.

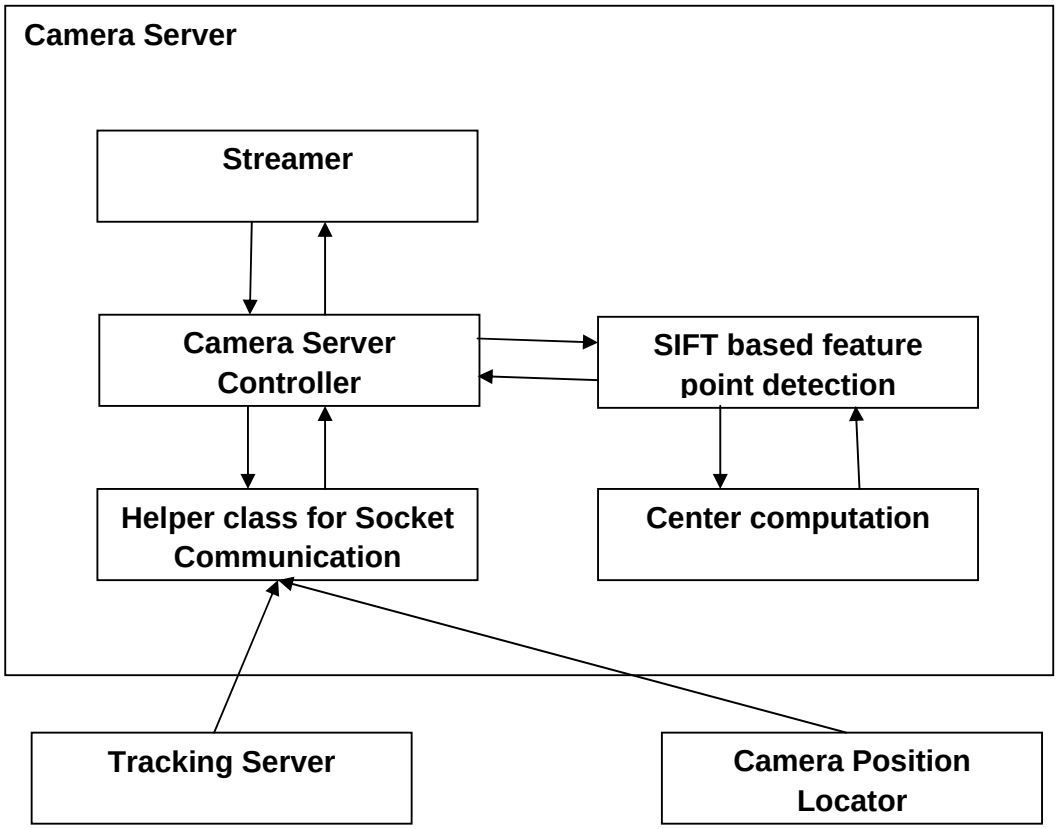

Figure 11 - Schematic diagram showing the functional modules of camera server

Architecture of the Automatic Camera Array Calibration System

Camera Position Locator is responsible for interactively displaying the location of the cameras in response to user adjusting the cameras. The camera position locator polls all the camera servers in the MiNT testbed requesting them to compare the feed from the camera with all the training images. Depending up on the location and orientation of images lying under each of the cameras, the automatic calibration system computes the position of each of the cameras and displays them in the global image of the testbed. All the three modules use the same Socket Helper module for socket communication. Also, the camera position locator does not poll the camera server with each training image one at a time for performance reasons. Instead, the camera server compares each image against the entire list of training image and returns a map containing the training image and the set of feature points found when matching it.

The returned result is pruned of outlying features and checked to see if there are at least a minimum number of feature points. If not, there is a good probability that only a part of the training image is lying under the current camera. In these cases, outlier elimination algorithm would not have been exact and it is better to ignore such images. The features are then used to compute the rectangular regions and then updated in the global image. It is to be noted that the global image is to cleared at the end of every cycle.

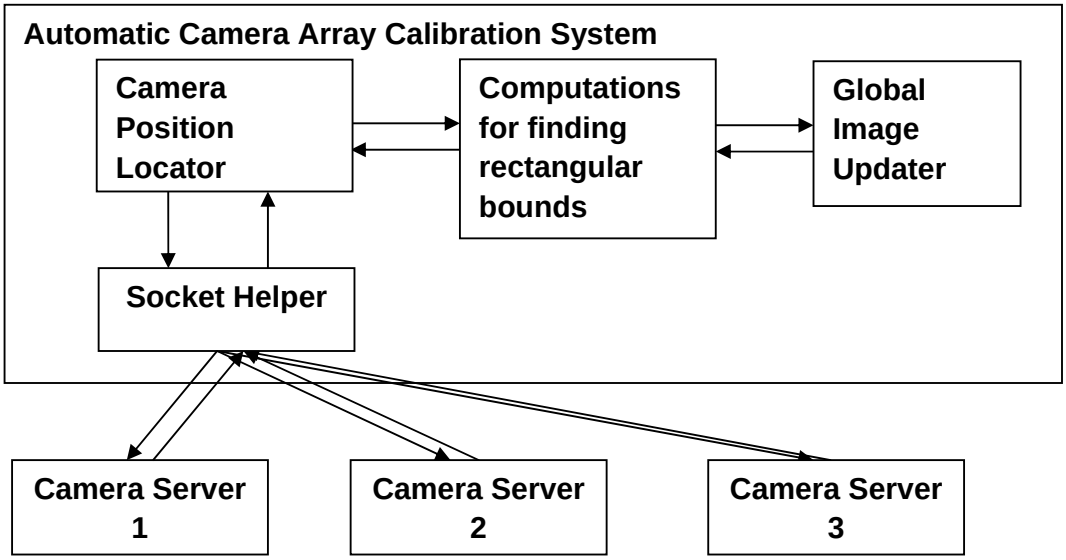

Figure 12 - Functional Software component diagram of Camera Position Locator

Architecture of the Tracking Server

Tracking server also makes use of the camera server. It communicates with all the camera servers to obtain the position of all the nodes under them. It then translates them to absolute positions making corrections for camera position and orientation. The tracking server also performs outlier elimination and intelligent tracking schemes to preserve accuracy and speed.

4.4 Automatic Calibration System Workflow

The actual manual steps involved in the setup of the testbed are detailed below:

Place images on the testbed at marked positions without any tilt. Positions of the marked positions are already measured before hand or the positions of the images are measured after they are placed on the testbed. If the set of images used for detection are different from the set of images used for tracking the images can even remain fixed to the floor when experiments are run. This way, whenever there is a change in camera position or new cameras are added to the testbed, their position can be automatically detected.

The cameras are attached to props looking down on the testbed. No special care needs to be taken to orient/align them. The automatic calibration system processes the image from each feed independently. In the initial run, it gives a list of cameras that do not have even two images under them. If the camera contains no image under it, the position of the camera is not displayed. However, if there is at least one image under the camera, the position of the camera is drawn with a default pixel to inch ration (typically 640 pixel/ 72 inches).

But given the density of images on the testbed and the fact that each camera covers a significant area, this is a rare occurrence. Should this arise, the user has to point these cameras in such a way that at least two images are captured by them. Once this step is done, the calibration tool generates an image of the testbed once, every 10 seconds. The cameras are represented as rectangles that show both the position and orientation of the cameras with respect to the testbed. Using this image, the user can verify whether the entire testbed is being covered or not. The image can be viewed in a browser with refresh timeout set to 10 seconds. This way user has live feedback whenever he moves the cameras. Since, the user has live feedback, it is easier for him to adjust the cameras to get a better coverage and better utilization of the camera area.

Whenever, the camera positions are changed, the position and orientation of the cameras are stored in a file. The tracking system takes this file as the input and automatically makes adjustments to the features coming in from different cameras based on their position and orientation. The system can be used very effectively even after the testbed has been setup. If new cameras are added or there have been small changes to the camera orientation, the testbed is still fully covered. The user can run the camera position locator tool and manually verify that the testbed is still fully covered. Once this is complete, he can start the tracking system and tracking will work seamlessly.

4.4.1 A Typical Camera Calibration Work Flow

We illustrate different possible scenarios faced during the automatic calibration process. It is to be noted that white represents the testbed dimensions, blue shaded, red bordered rectangles represent the area covered by the cameras. Initially, it is assumed that all the cameras are highly skewed with some cameras facing the testbed at an angle.

Initially, one of the camera is facing away from the testbed. It does not even contain a single image under it. In this case, the camera calibration system prompts the user to adjust the camera and displays the positions of the rest of the cameras. The image generated by the automatic camera calibration system when the top left camera is facing away from the testbed is shown in Figure 13. In *Figure 14*, this camera is slightly adjusted to have one image under it. Though the automatic calibration system still prompts the user for the particular camera to be adjusted to have atleast two images under it, it computes the position and orientation of the testbed and displays the camera using default values for pixel-to-inch ratio. This gives the user, a sense of where the camera is in relation to the testbed. It can be seen in *Figure 14*, that the top left camera is considerably smaller than the other cameras. This is because the default pixel-to-inch ratio is very conservative and is just meant to give a sense a location to the camera.

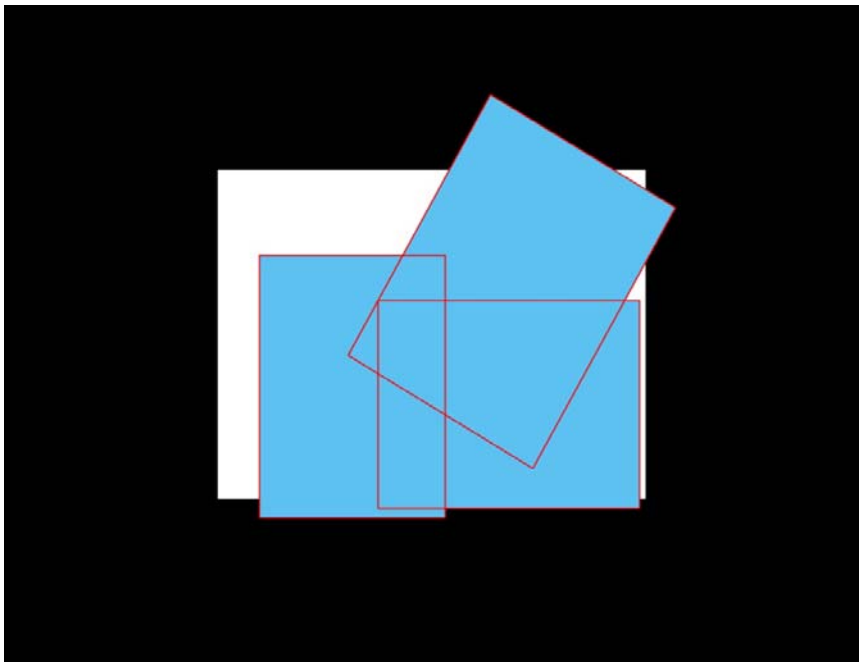

Figure 13 – Figure illustrating steps in Automatic Calibration. Cameras not aligned. One camera without any image under it.

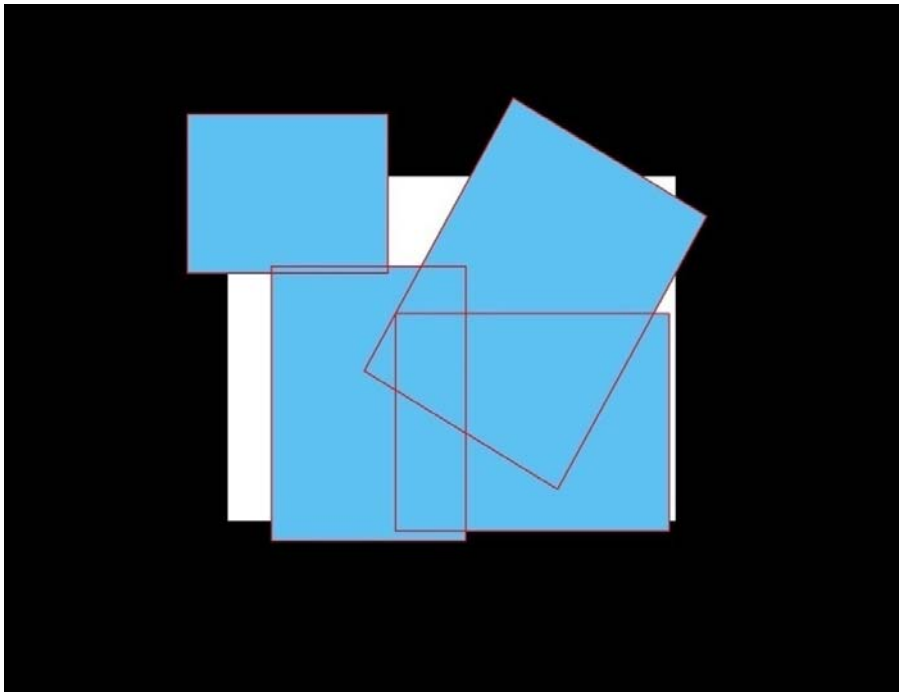

Figure 14 – Figure illustrating steps in Automatic Calibration. Cameras not aligned. Top left camera has a single image under it.

Following this the user further adjusts the camera to face the testbed. This time, the camera is facing the testbed at an oblique angle .The remaining

cameras are also adjusted to fully cover the testbed. Portion of the testbed as seen from an obliquely pointing camera can be seen in Figure 15. Following this, a final image of the testbed is generated that fully covers the testbed, Figure 16. It is to be noted that even with this amount of irregularity in camera position and orientation, tracking still works reasonably.

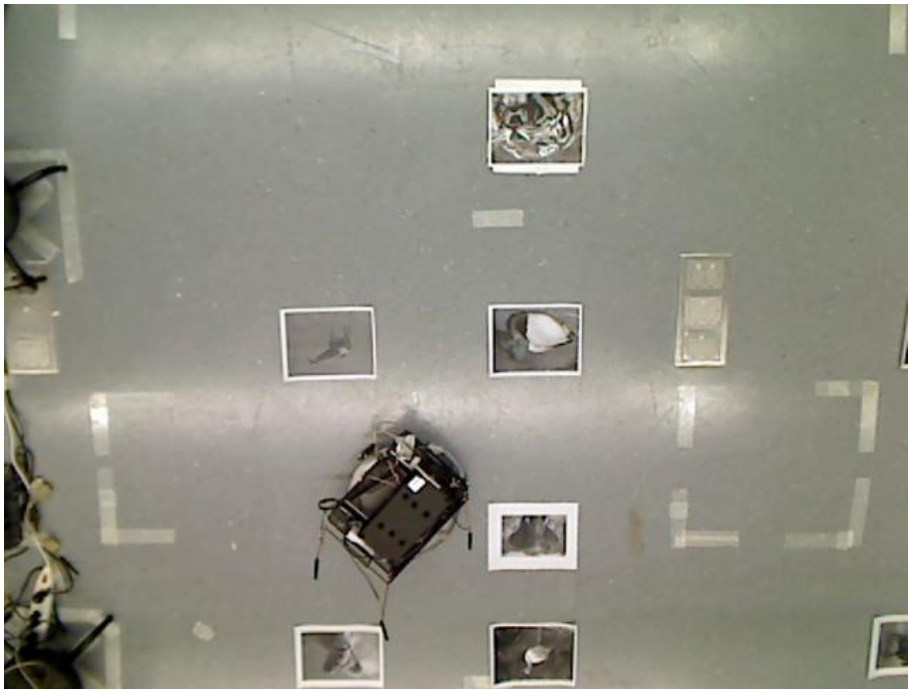

Figure 15 – Figure showing a testbed camera obliquely pointed to the tesbed.

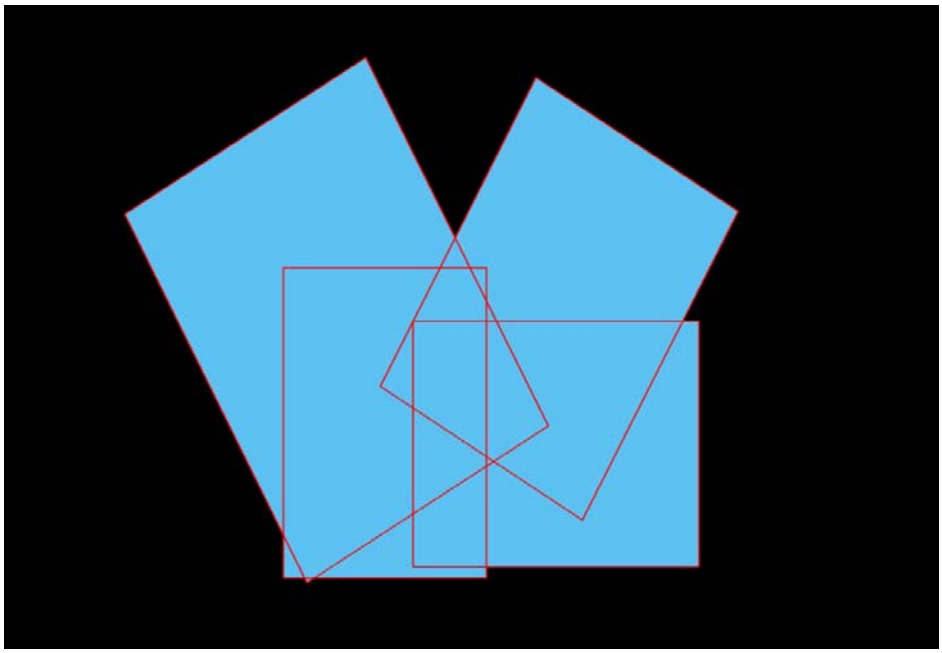

Figure 16 – Image generated by the automatic calibration tool showing all the cameras fully covering the testbed.

4.5 Implementation Issues

We faced a host of implementation issues during implementation and integration of the Robot tracking system and the Automatic calibration System. Most of the problems were due to the fact that the system involved inter-working and integration of lot of components. However, on a positive light, these issues made the design of the system more extensible and more scalable.

4.5.1 USB Bandwidth Problem

USB cameras typically reserve a percentage of the available USB controller bandwidth [8]. This happens irrespective of the frame-rate. As a result of this fact, even though we require a low frame-rate, typically 5 frames per second, more than 3 cameras cannot be connected to the same USB controller. One solution to this problem is to use 1 USB adapter card for each camera, effectively one controller for each camera. This way, there will no contention for USB bandwidth of USB controller. As a result of this design

issue, camera servers were designed to operate on one camera at a time. Each server can run as many camera servers as the USB bandwidth of the server would support.

4.5.2 Incompatible Drivers for Camstream

We had to buy new cameras for the testbed as the old cameras in the testbed were found to be noisy. However, the new cameras Logitech Quickcam Pro 9000 were compatible only with V4L2 which were only available in the later versions of linux kernel [9]. Particularly, they were not compatible with the existing pwc drivers. But when the tracking servers were upgraded to 2.6 kernel, we discovered that camstream, the open source software that was customized for tracking, would not work with Quickcam Pro 9000 though it supported V4L2 [10]. This forced us to adapt aV4L2 based video grabber software, streamer which was used for tracking. As a result, the new robot tracking algorithm and the tracking server are both designed to be completely decoupled from any frame grabbing software used. This would make the shift to a new camera frame grabbing software with minimal changes to code.

4.5.3 Distributed System Architecture

Designing and implementing a system composed of several distributed components caused unique problems in design and debugging. The design revolves around asynchronous requests from camera position locator and tracking server to camera server and asynchronous responses. To keep the asynchronous behavior under acceptable limits, the design tries to batch requests and responses whenever possible. This is a desired trait as some components of the testbed like the tracking server and the automatic calibration system require the position information of the entire testbed at a snap shot in time.

To achieve this behavior, all centralized components like tracking server and the automatic calibration system queue responses from multiple camera server, and only process the responses when all of them have responded. This approach has a caveat that if one camera server dies, the

entire testbed becomes useless. To handle this case, we timeout after a certain time interval and if the same camera server misses updates continuously for a pre-configured number of times, the camera server is marked non-functional. As of now, though the current implementation detects this case, and points out the erring camera server, the process does not continue. This because computing trajectory in a testbed with failed cameras located arbitrarily is difficult.

Chapter 5

Evaluation

The two main contributions from this thesis are developing a SIFT based robot tracking system and camera array automatic calibration system. The first step of evaluation was to study the performance of the different stages of the SIFT based image detection algorithm, evaluate its accuracy and finally the setup time and accuracy of Automatic camera array calibration system.

5.1 Performance Evaluation of SIFT image detection algorithm used in the testbed

First, we study the performance of SIFT algorithm for generating and matching features. The SIFT algorithm as used in the image detection system has three main components that contribute to the total running time of the algorithm. They are

1. Generating feature points for training images.
2. Generating feature points for the image captured from the camera.
3. Time taken for matching features.

5.1.1 Generating Feature Points for Training Images

This step is executed once at the beginning of the setup. Once features are computed for all the training images, this data is preserved for the duration of the setup and experiment. The time taken for this step is therefore non-recurring but proportional to the number of training images in the testbed. But since the training images are smaller in dimension than any of the video frames captured in the testbed, the time taken for this step should be insignificant.

From the above Table 1, it can be clearly seen that the time taken for detecting all the features in a given image is dependent more on the number

of features present and less on the size of the actual image. This fact is easily illustrated by comparing the first image and last image.

Image size	Number of Features	Time taken in ms
80x66	31	27.697
78x72	40	33.335
75x62	66	34.694
82x58	73	47.905
75x61	75	64.133
80x60	95	71.093
Average	63.33	46.476

Table 1 - Comparison of times for generating features for different images sizes and number of features

5.1.2 Generating Feature Points for the Image Captured from the Camera

From the above table, it should be clear that with the increase in the number of nodes under a camera, the number of features within them will increase and hence the time taken for computing the number of features.

Number of Images in testbed	Number of features in camera image	Time taken in ms
0	89	456.809
1	136	486.734
2	227	367.07
3	290	541.801
4	367	576.086
5	445	644.22
6	533	654.313
Average		**532.4333**

Table 2 – Comparison of time taken for generating features for an image taken in the testbed.

The values that we obtained in table 2 are in accordance with the fact that the computing time slightly increases with the increase in the number of feature points. The first entry was taken with an empty but no means clean testbed. The testbed had several tape measurements and measurement markings. Even with 89 features for the background noise in the testbed,

localization was pretty accurate for all the test cases. Also, it has to be noted that even with 6 images lying under the camera, a very unlikely occurrence for both tracking server and the automatic configuration utility, the time taken for processing the image was less than a second. It is to be noted that since this operation is performed by the camera server, there is no increase in overall delay irrespective of the number of cameras as long as camera servers are load balanced in multiple servers.

5.1.3 Time taken for Matching Features

Since the image position and orientation are determined by matching the features from training image to those from the actual image, time taken for matching will be proportional to the number of features present in the camera frame. This means that the time taken for matching would also increase if there are more nodes present within a single camera.

Number of Images in testbed	Number of features in camera image	Time taken in ms
0	89	2.026
1	136	3.146
2	227	5.55
3	290	6.573
4	367	27.656
5	445	10.655
6	533	12.692
Average		**9.756857**

Table 3 – Comparison of time taken for matching feature points of an image with training features.

From table 3, It can be seen that the time required for matching SIFT features is insignificant compared to the time taken for generating the features in the image. However, as already predicted a small increase in the time taken is noticed as there is an increase in the number of features. This cost increases linearly with the number of nodes in the testbed. This also shows that though the automatic calibration system compares each frame captured from the camera with all the training images, the comparison should take negligible time, given that the Automatic calibration system updates its global image only once every 5 or 10 seconds.

5.1.4 Evaluation of the Accuracy of the SIFT Detection Algorithm

We also need to evaluate the accuracy of the SIFT image detection algorithm. We measure both the accuracy in the position and orientation of different images when they are present with different orientations.

The table below (Table 4) is constructed from data taken from 5 different frames of the testbed with two images dog (image with 24 feature points) and tiger (image with 72 features) placed at different positions of the testbed. However, since the orientation cannot be precisely controlled on the testbed, the five images are then digitally rotated in increments of 45 degrees to obtain eight different orientations. This step is performed only to enable precise measurements of the error in position and orientation. The error value for each image in each orientation is the average of the five error values at each specific orientation (say dog at 45 degrees is the error computed from 5 different positions of the dog in the testbed). From the above table, it can be seen that the accuracy increases with the increase in the number of features, but only marginally. From our experiments, we determined that accuracy remains acceptable when the number of number of feature points is greater than 20. So we fixed the threshold value for the detection of the image at 20 feature points. Since the pixel-to-inch ratio in the testbed is 7.619, the average error in position is only 1.204 inches.

Orientation	**Features**	**Error in position in pixels**	**Error in orientation in degrees**
0	24	5.59017	0
45	24	7.905694	+0.800
90	24	5	+0.250
135	24	10.30776	+2.4
180	24	12.5	+0.6
225	24	15	+1.8
270	24	10.6066	+1.0
315	24	19.52562	+0.85
Average		**10.8044**	**0.9625**
0	72	5	0.100

45	72	2.5	+0.5
90	72	9.002	+0.5
135	72	5	+1.0
180	72	9.013878	-.05
225	72	12.5	-2.0
270	72	10.30776	-0.3
315	72	7.071068	-1.8
Average		**7.549**	**0.71875**

Table 4 - Comparison of the accuracy of SIFT algorithms for different orientations and images with different number of features

5.2 Evaluation of the Camera Array based Automatic Calibration System

Since the primary goal of this thesis is to decrease the time of setup, one of the main evaluation parameter for automatic calibration system is the time taken for the setup. The time taken for the setup of the old test bed and the new SIFT based testbed consists of the following factors:

1. Time taken for selecting SIFT images
2. Time taken for assembling the MINT-m nodes.
3. Time taken for physical testbed setup
4. Time taken for camera adjustments for alignment
5. Time taken for color based adjustment
6. Time taken for setting up computing resources and software.

The improvements propsed in the thesis reduce factors 1, 3 and 4. Time taken for assembling MiNT-m nodes is proportional to the number of nodes in the testbed.

Operation	**MiNT-m Testbed**	**SIFT based Testbed**
Time for Selecting Images	-	30 mins 0 if images are published
Time taken for physical testbed	1 hour	1 hour

Time taken for camera adjustments	1 hour/camera	45 minutes
Time taken for color based adjustments	2 hours/camera	-
Total	**1 hour + 3 hour/camera**	**2 hours 15 minutes**

Table 5 - Time taken for setting up MiNT-m Testbed vs SIFT based testbed

From the above table, it becomes clear that the redesigned SIFT based testbed takes significantly less time to setup compared to the old MINT-m testbed. Also, it has to be noted that the MINT-m design does not scale to more than tens of robots.

5.3 Evaluation of the Accuracy of the Camera Array based Automatic Calibration System

Accuracy of Camera Position Locator was also experimentally evaluated initially with 2 images placed under a camera and then with 3 images. The above experiment was also run for 8 different digitally rotated versions of the camera image to closely monitor the orientation and an average of 5 values were taken.

Orientation	**2 images under the camera**		**3 images under the camera**	
Orientation	**Position**	**Orientation**	**Position**	**Orientation**
0	1.224	0.4	1.08234	0.3
45	5.512769	0.4	5.24616	0.2
90	11.42503	-0.8	10.7132	-1.0
135	16.05557	+1.0	13.7619	-0.2
180	18.31154	+0.2	17.6956	-0.7
225	16.49337	+1.8	14.2164	0.8
270	11.67262	-0.8	10.0051	0.8
325	5.81082	-0.9	4.8674	-0.5
Average	**10.81321**	**0.7875**	**9.698513**	**0.5625**

Table 6 – Comparison of accuracy of camera position locator for different orientations and images under the camera.

As in the above experiment, though there was a slight increase in accuracy, the increase was only slight. So we can conclude, that we can get the required amount of accuracy even if we have 2 images under each camera.

Chapter 6

Conclusion

6.1 Contributions to the thesis

In this section, we highlight our contributions to the MiNT-m testbed and advantages of these new techniques. We also discuss possible improvements to the different modules. We replaced the color based node location system with an image recognition system based on SIFT. Since, the SIFT algorithm is designed to match images, the testbed can be scaled to any number of nodes. This is also made easy by the flexible design of camera server which allows each server to handle different number and sets of cameras. The characteristics of the SIFT algorithm along with our algorithm to eliminate outliers has drastically improved the detection accuracy of the testbed. We also developed tools to make the process of finding desirable images for SIFT easy. Also, a large set of images could be published along with the software, to make the setup easier. Another tool was developed to make the process of setting up cameras easier. Since cameras are mounted on rods supported by props, setting up the testbed (The camera alignment part) takes very little time as the camera position locator tool accurately displays the position of the cameras in the testbed in real-time.

6.2 Future Enhancements to the SIFT Image Selector Tool

The tool is used to make the process of selecting images to be used for node localization easier. The entire process of choosing images can be automated by automatically crawling for images of particular size and type in a public image site given the number of images required. The current implementation just returns the set of all usable images. It would be more useful if it ranks all the returned images by desirability as the number of images required is typically less than the images required.

This procedure is basically an offline procedure. The algorithm for selection of images can be made much more efficient if for each training

image, we were also allowed to eliminate only those feature points that overlap with a number of other images. However, for this to work, the selected feature points should be persisted and read in by the algorithm for the tracking system. One disadvantage of this approach is that there would be reduction in image detection accuracy if many such features are dropped.

6.3 Future Enhancements to the Camera Array Automatic Calibration

A simple enhancement to increase the update frequency would be to remember the training images that were detected under the camera during the previous scan. After this cycle, it makes sense to only compare those images which are a fixed distance away from these images.(say 1.5 m). With this enhancement, the update frequency can be as high as 1 or 2 s and since a camera can be physically moved only such a distance in this time duration, it makes sense.

Instead of the user determining if the complete testbed is covered, the system can automatically determine if the testbed is covered or not. This can be easily achieved by shading each rectangle with a color instead of just drawing the boundary. Once all the cameras are processed, the testbed is scanned once more to determine if all the pixels in the testbed are shaded. If not the testbed is not fully covered. This enhancement was not completed mainly for performance reasons.

Given which portions of the testbed are left uncovered, the testbed could suggest cameras that can be moved to cover the overlap. However, this is a non-trivial task since given an uncovered area and several cameras occurring around that area, there are several ways of filling up the free space. Also, given the availability of network controlled cameras which are capable of pan/ tilt, the output from the previous step can be fed to the cameras and they can tune themselves. However, the currently available cameras with pan/tilt facility are costlier than normal cameras.

Bibliography

[1] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu and M. Singh, "Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols", Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities, 308 - 309, February 2009.

[2] D. Johnson, T. Stack, R. Fish, D.M.Flickinger, L. Stoller,R. Ricci, J. Lepreau, "Mobile Emulab: A Robotic Wireless and Sensor Network Testbed", INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings, 1-12, April 2006.

[3] Lars Cremean , William B. Dunbar, Dave van Gogh, Jason Hickey, Eric Klavins, Jason Meltzer and Richard M. Murray, "The Caltech Multi-Vehicle Wireless Test bed", Proceedings of the 41st IEEE Conference on Decision and Control, 2002, 86-88 vol.1, Dec. 2002.

[4] C. Mitchell, V.P. Munishwar, S. Singh, Xiaoshuang Wang, K. Gopalan, N.B. Abu-Ghazaleh, "Testbed design and localization in MiNT-2: A miniaturized robotic platform for wireless protocol development and emulation", Communication Systems and Networks and Workshops, 2009. COMSNETS 2009, 1-10, Jan. 2009.

[5] A. Arora, E. Ertin, R. Ramnath, M. Nesterenko, W. Leal, "Kansei: a high-fidelity sensing testbed", Internet Computing, IEEE vol 10;issue 2, 35- 47, Mar - Apr 2006.

[6] Wikipedia page for QR codes http://en.wikipedia.org/wiki/QR_Code

[7] D. Lowe, “Distinctive image features from scale-invariant keypoints”, International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

[8] Problem of USB bandwidth

http://www.lavrsen.dk/twiki/bin/view/Motion/FrequentlyAskedQuestions#How_do_I_get_Motion_to_work_with

[9] V4L2 included in kernel in 2.5.x/2.6.x

http://www.linuxtv.org/wiki/index.php/Development:_Video4Linux_APIs

[10] UVC incompatible Software

http://www.quickcamteam.net/software/linux/v4l2-software/uvc-incompatible-software?searchterm=camstream