# DISTRIBUTED CALIBRATION OF A CAMERA SENSOR NETWORK

By

VIMAL MEHTA

Bachelor of Engineering in Electronics and Communications
Andhra University
Visakhapatnam, Andhra Pradesh
2005

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2008

DISTRIBUTED CALIBRATION OF A CAMERA

SENSOR NETWORK

Thesis Approved:

Dr. Weihua Sheng

Thesis Advisor

Dr. Qi Cheng

Dr. Sohum Sohoni

Dr. A. Gordon Emslie

Dean of the Graduate College

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

# CHAPTER 1

# INTRODUCTION

In this chapter, we present the motivation behind our work in section 1.1. Section 1.2 of this chapter throws light on the need for wireless camera sensor networks and its potential applications. Section 1.3 presents an overview of the state of the art calibration techniques that are in use. The organization of the remaining document is presented in section 1.4.

## 1.1 Motivation

The advancements in technology and the emergence of a variety of sensors have provided potential research options in the field of wireless sensor networks [24]. This has led to the wide use of wireless sensor networks in various applications involving data acquisition, sensing and control. The emergence of new sensors has paved the way for the development of a variety of effective, low power and low cost vision based sensing platforms [19], [21], [22], [25], [27], [29]. Such platforms of wireless camera sensor networks play a crucial role in a variety of applications ranging from a habitat monitoring, virtual tour of an environment and intruder tracking in defense [32]. A wireless camera sensor network as the name indicates is a collection of nodes, connected to one another using wireless communication. Each node of the network is equipped with a camera. Such networks not only provide rich image information but also facilitate

motion sensing with high fidelity at low powers [21]. Calibration is the process of determining the internal parameters like focal length, distortion, skew coefficient and external parameters like position and orientation of a camera in an environment. Calibration is required to correct the errors caused due to device imperfections and aging [33]. With calibration, a camera sensor node can maintain up to date information about its location and the orientation in an environment. A wireless camera sensor network operates either as a centralized or as a distributed network. In the centralized mode, all the nodes of the network communicate their data to a central node. The central node runs computational algorithms on the received data and sends back the results to the respective nodes. In the distributed mode, each sensor node runs computational algorithms on its own processor and may also exchange its results with its neighboring nodes. In the case of a large scale sensor network operating in a centralized mode, data transmission to a central station by the other nodes increases the transmission overhead and consumes time. It may also increase the power consumption at the central node at the expense of the remaining nodes of the network. Failure of the central node will lead to the failure of the entire network. Distributed camera calibration [1], [16], on the other hand, has the advantage that a node need not depend on other nodes for its calibration. A distributed node can calibrate by itself and communicate its position and orientation information to its neighbors. The sensor nodes perform complex mathematical calculations to carry out the calibration on their own processors. However, the growing need for low power and low cost sensing technology has limited the computational ability of sensor nodes. Hence, there is a need to develop lightweight calibration algorithms for the purpose of configuring and operating these distributed sensor nodes [21] so as to employ them for

potential applications. The main aim of this thesis is to develop a simple and lightweight distributed computational algorithm to calibrate a camera sensor network

## 1.2 Problem statement

Consider a set C of n camera sensor nodes, connected by a wireless network that is randomly deployed in a battlefield for the purpose of intruder detection. As the battlefield is an unknown environment, a node has no information about its coordinates and orientation. If there is a detection of enemy intrusion by a sensor node, it will not be able to determine the location of the intruder. So, in order to localize the intruder, the sensor node must have prior information about its own position and orientation. Calibration techniques facilitate this purpose. In order to calibrate itself, a sensor node must know the position coordinates of certain detected target. This implies that a localized target must be present in order to calibrate a sensor node. This leads to three main issues. Firstly, a detected enemy target is non-cooperative and hence will not assist a sensor node in its calibration. Secondly, a reference coordinate system is required to localize the target and calibrate the sensor nodes of the network. Thirdly, the calibration algorithm must involve simple mathematical calculations so that it can be implemented in a distributed fashion with ease on low power processors having limited computational capabilities. In a distributed network, the nodes can communicate the tracking information n with one another without the need to send the data to the central node. This increases the speed at which the tracking responsibility can be transferred from one node to another node in the network [14]. As no image data is shared in the network, the amount of data transferred

by a node is relatively less, which not only reduces the power consumption of a node but also increases the life-span of the network.

So, there is a need for a simple calibration algorithm in which a cooperative friendly target (for example a soldier) can move around in the environment, collaborate with the camera sensor nodes and assist them in distributed calibration. To localize the target, a few nodes of the network work simultaneously to establish a reference coordinate system (the reference frame). The cooperative target can communicate with the sensor nodes in the battlefield using the same wireless network connecting all the nodes. The moving target must have a mechanism to update its position coordinate information from time to time. Use of a Global Position System (GPS) can aid this process. However, GPS provides accuracy in the order of meters. This is undesirable in small or indoor environments. Also, the GPS signal is not available in all types of environments and hence there is a need to search for position sensors that can be deployed for small and indoor scenarios. Use of a wearable position sensor module is one solution. The target is equipped with a wearable position sensor that can be mounted on its body. The sensor module is equipped with an accelerometer, a gyroscope and a compass. Whenever the target moves, the readings from all the three sensors are combined using dead reckoning to calculate the distance moved and the heading from the previous position. This information helps the target to update its position coordinates from time to time in the environment. The target while moving in the environment, exchanges its location information with camera sensor node it encounters in its path. This helps the sensor node to calibrate itself and hence determine its position and orientation in the reference frame. The aim of the calibration algorithm is to find the pose $(x_i, y_i, \theta_i)$ of a sensor node $i$,

where $(x_i, y_i)$ specify its location in the reference coordinate system and $\theta_i$ is the orientation of its camera with respect to the positive x-axis of reference coordinate system. In this thesis, we try to solve for the calibration problem only in 2-dimensional environments.

## 1.3 Related work

Camera calibration is the process of determining the internal parameters of the camera like its focal length, principal point, skew coefficient, distortion and some external parameters like its location and orientation in the environment. This section describes the current state of the art calibration techniques that are being employed. Camera calibration has been under research since quite a long time in the photogrammetry [17], [18] as well as in the Computer Vision community [12], [20] and [28]. Photogrammetry is a technique which uses images of an object to determine its geometric properties. Two or more images of the object are taken at different orientations and feature points common to all the images are identified. Vectors joining the centre of the camera to each of the identified points are constructed and triangulation method is used to determine the 3-dimensional coordinates of a required point on the object. This approach however requires an expensive experimental setup. Self calibration is a technique that relies on the motion of a camera in a static scene. It does not make use of any object to calibrate a camera. The camera captures a number of images of a static scene while it is moved relative to the static scene. The rigidity of the scene from the captured images provides constraints on the internal camera parameters. With fixed internal parameters the

correspondence between at least three of the captured images is sufficient to recover both the internal (focal length and distortion) and external parameters [13].

A few methods [15], [13] use landmarks at known coordinates to calibrate a camera. The projection of the image of these landmarks on the lens of the camera is combined with the principle of optics to determine the position and orientation of a camera. In [13], Zhang proposes a flexible camera calibration technique between photogrammetry and self calibration. In this method, a planar pattern is shown to a camera at two or more different orientations. The camera captures the images of this pattern at each orientation. From the captured images, feature points are extracted and calibration is done by estimating five intrinsic and all extrinsic parameters of the camera. The parameters are iteratively refined using a maximum likelihood criterion.

In [15], Rekleitis *et al.* proposed a technique for automated calibration of a camera sensor network using landmarks. In this technique, a world grid coordinate system is established in an indoor environment. One or more cooperative mobile robots carrying a calibration pattern move around in the environment A coordinate system with its origin at the center of a the robot is assumed. The transformations from the world grid coordinate system to the robot centered coordinate system are predetermined. The mobile robot stops, whenever it passes through the field of view of a camera. A number of images of the calibration pattern are taken by the camera and are used to determine the internal and external parameters of the camera. These parameters are measured with respect to a camera centered coordinate system. The camera communicates these parameters to the robot which finally transforms the internal and external parameters to the world grid coordinate system. The above two methods, however require the use of landmarks or

patterns to carry out calibration which are not available in a real world situations like remote monitoring of natural habitats or battlefields.

To overcome the above constraint of using landmarks, in [14], Kulkarni *et al.* propose an approximate initialization technique to determine the relative locations and orientations of camera sensors without the use of landmarks. In an environment with a network of distributed cameras, this technique determines the degree of overlap and the region of overlap of each camera sensor node with the other camera sensor nodes of the network. These parameters help in achieving a desired probability of event detection and maintain reliability in tracking a moving object. A reference object of known size at random points in the environment. The camera sensor nodes are programmed to capture images of this reference object on a duty cycle basis. The nodes estimate the degree of overlap with various sensor nodes by processing their respective images and determining the reference points in their field of view. By determining a subset of reference points that are visible to two or more cameras, the region of overlap between the cameras is given by the union of cells containing the reference points visible to each camera. The region of overlap of the camera is used to estimate the target path, and hence provide reliable information in estimating the next sensor node which has to take over the tracking responsibility. Assuming that the origin of a reference frame is located at the center of the sensor node, simple optics is used to determine the distance between the camera and the reference point. As the size of the object as well as the values of the internal parameters is known, the sensor nodes can calculate the orientation of the reference point with respect to their center. However, this calibration technique only helps in determining the relative locations of the reference point and the orientations of the camera sensors. In this

technique, the tracked target is not localized with respect to a single measurement frame but is localized with respect to the reference frame of each camera. To simplify tracking, many applications require the measurements for target localization and camera calibration to be carried out in the same frame. This can be achieved only when the target and camera sensors are present in a single reference frame.

A single reference frame based calibration technique is presented in [1]. In [1], Lee and Aghajan present a collaborative technique to localize the nodes of a camera sensor network using opportunistic observations of a target. Each node uses lightweight in-node image processing to extract the coordinates of the center of the blob of an object it detects. A sensor node that detects an object sends trigger signals to its neighboring nodes. A neighboring node, which also detects the object, forms a reference coordinate system with the triggering node. The triggering node and its neighbor (also called the helper node) act as reference nodes. The reference nodes along with more than one uncalibrated node can participate in tracking the object. Each node uses a pin-hole camera model to determine the angle between the optical axis of its camera and the line joining the center of its camera to the center of the detected blob. All the tracking nodes collaborate with one another and exchange the tracking information. Every node then uses Gauss–Newton method [34], [35] on the exchanged information to determine its position and orientation in the reference frame. However, this calibration algorithm relies on the assumption that all the reference nodes and the uncalibrated nodes that are participating in the tracking must view the target simultaneously. In order to perform the calibration, this method requires a minimum of three sensor nodes and at least five observations of the target be taken by each sensor node.

In [16], Funiak *et al.* addressed the camera calibration by solving a distributed simultaneous localization and tracking (SLAT) problem [15]. The SLAT problem is solved by taking images of a moving target and obtaining a joint distribution over the moving target locations and the poses of the cameras in an environment. As these are continuous variables, it is very difficult to represent a general distribution using these variables. Hence, a relative over parameterization (ROP) is used to represent the complex distributions as a single Gaussian which can be solved by using simple matrix operations. The nonlinearity and uncertainty associated with the camera angle always lead to inaccurate SLAT results. So, the ROP is combined with conditional hybrid linearization using a Kalman filter to provide precise SLAT solutions. The proposed technique to solve the SLAT problem, involves nonlinearity not only for the target localization but also for the camera calibration problem. In this document we present a cooperative target based distributed camera calibration. Unlike in [12] and [15], this technique does not require a landmark pattern to calibrate a camera and hence is applicable to real world situations. Due to its simplicity, the proposed calibration algorithm can be implemented on low power camera sensor nodes. In [20], Liu *et al.* propose an automated wireless self calibration protocol for camera sensor networks. This work combines the principles of Computer Vision, optics and vectors with the internal parameters of a camera to estimate the location and orientation of the camera. The work assumes that a device equipped with an ultrasound based positioning sensor moves around the environment and generates a set of reference points in the field of view of each camera that it encounters. To perform calibration the camera is provided with the location information of four reference points in its field of view by the moving object. The sensor node then determines the vectors

joining its centre to each of the reference points. It then employs a non-linear solver to estimate the location of the camera. After determining its location, the sensor node uses a subset of three reference points to determine the orientation of its camera. The obtained external parameters of the camera are iteratively refined by using additional reference points in the field of view of the camera. The external parameters are now used to determine the region of overlap which helps in localizing and tracking the moving object.

Though the idea behind our work looks similar to the work proposed in [20], it is different in several ways. The first difference is that the calibration algorithm in [20] requires four known locations in the field of view of the camera only to determine its location. After calculating its coordinates, the sensor node uses them with a subset of three locations and combines them with the principles of optics to determine its orientation. In the distributed calibration algorithm proposed in this thesis, with knowledge of just three distinct locations in its field of view, the camera can determine the coordinates of its location and its orientation angle simultaneously. The second difference is that in [20], the authors assume that the moving target is equipped with an ultrasound based positioning system while in our work; we assume that the target is equipped with a dead reckoning based positioning system. The third difference is that the work in [20] uses an estimation process to perform calibration and as such there is need to provide initial guesses for the location and orientation of the camera. If the guesses are not properly selected, there is always a chance that the computation may get stuck at a local minimum which is not desirable. Our work on the other hand does not require any initial guesses to perform calibration and hence rules out the possibility of getting stuck at a local minimum.

## 1.4 Organization of the thesis

Chapter 2 of this thesis presents an overview of the calibration algorithms. Chapter 2 brings out a brief comparison between existing calibration techniques and the proposed calibration technique. It also discusses the system model for distributed calibration and the mathematical calculations involved in the calibration algorithm. The proposed calibration algorithm is practically tested on real camera sensor network containing three CMUCAM2 cameras. The experiment procedure and an analysis of the results are discussed in chapter 3. The proposed distributed calibration algorithm is simulated using a large scale camera sensor network in chapter 4. The Player/Stage robotic simulation software used to carry out the simulation, the simulation procedure and a statistical analysis of the simulation results are also discussed in chapter 4. The conclusions along with the scope for future development are presented in Chapter 5. A detailed description of the VB.NET application used in the practical experiment described in chapter 3 is also discussed in the appendix A. The procedure for the simulation of the calibration algorithm using Player/Stage is explained in detail in appendix B.

# CHAPTER 2

# BASIC PRINCIPLE OF DISTRIBUTED CAMERA CALIBRATION

In this chapter, we propose a novel method for a distributed camera calibration using cooperative target observations, which is suitable for low power applications. In this chapter we also discuss the assumptions used for the proposed calibration algorithm followed by a mathematical analysis of the calibration technique.

## 2.1 Basic idea

The calibration technique proposed in [1] requires simultaneous target observations by uncalibrated nodes. The non-linear system model discussed in the paper [1] requires a minimum of three camera sensor nodes to take at least five target observations. We propose a calibration technique in which two reference cameras simultaneously track and localize a cooperative target. The cooperative target uses dead reckoning on the position sensor readings so as to update its current location coordinates, while moving in the environment. This localized target while wandering through the environment collaborates with the sensor nodes and assists them in their calibration.

## 2.2 Assumptions

In this algorithm, we assume the target is cooperative and can communicate with the sensor nodes in the environment using wireless connectivity. The cooperative target is assumed to be equipped with a wearable position tracking sensor. This tracking sensor informs the target about the size and heading of each step its makes during its motion. This assumption is similar to one of the assumptions in [20]. Once the cooperative target is localized in a reference coordinate frame, the moving target uses dead reckoning method [31] to maintain accurate information about its current location in the reference frame.

## 2.3 Working of the system

The proposed system consists of wireless sensor nodes communicating with one another over a wireless network using the zigbee protocol [23], [36]. Each wireless node is interfaced to a camera sensor. The camera is equipped with image processing software to extract the center of the detected blob. Thus, the camera can extract the locations of the center of the detected blob with respect to its field of view.

The camera sensor nodes are deployed at unknown locations and orientations in an environment. A cooperative target, a human soldier, equipped with a dead reckoning sensor, wanders randomly in the environment. It is assumed that the soldier can communicate with sensor nodes over the same wireless network which connects all the other camera sensor nodes present in the environment.

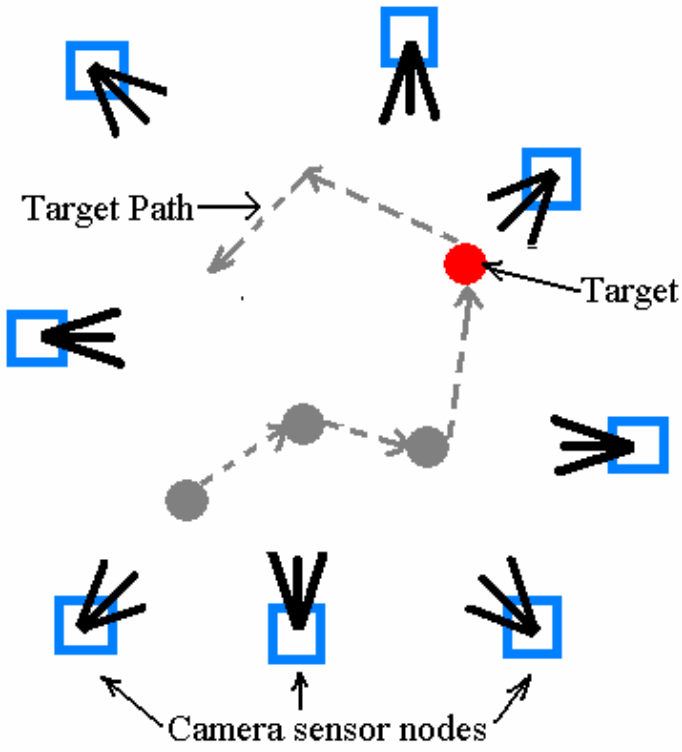



Figure 2.1: An example for a camera sensor network

Whenever a node detects the moving target, it sends out trigger signals across the network. On receiving the trigger signals, other sensor nodes in the network start acquiring frames. One of the nodes that can track the target is designated as a helper node. The triggering node and the helper node together form a reference coordinate system. Each of the reference nodes captures the target's image and extracts the center of the detected target's blob. For every captured target frame, the sensor nodes determine the relative angle between optical axis of their camera and the line joining their camera centers to the center of the target. This information is exchanged among the reference nodes and is used to estimate the current location of the target in the reference frame. Once the reference nodes perform localization, they inform the target about its current location in the reference frame. From here, the target uses dead reckoning to update its location information during its motion. The localized target now wanders in environment

to calibrate the remaining sensor nodes. Whenever the target falls in the field of view of an uncalibrated camera, the camera extracts the coordinates of the center of the detected blob and simultaneously requests the current location coordinates from the target. In a similar fashion, the sensor node extracts the blob center of the target at three different locations in the field of view of its camera. At each of the three distinct locations, the pin-hole camera model is used by the node to determine the angle between the optical axis of its camera and the line joining its camera center to the center of the target. The camera sensor node performs simple mathematical operations on the blob and location information of the target to determine its current position and orientation in the reference coordinate system. The sensor node also communicates its location and orientation information back to the wandering target. Thus, the localized target while moving in the coverage area of the network helps the camera sensor network nodes to calibrate themselves and also keeps a record of the locations and orientation of all the nodes it encounters in its path.

## 2.4 System model

The system model to implement the proposed calibration algorithms is shown in Figure 2.2. The reference coordinate system is formed by taking two nodes of the network as reference nodes and placing them as shown in Figure 2.2.

The origin of a reference coordinate system is assumed to be located at the first reference node and the second reference node is placed at a unit distance from the first node. In this system model, the reference nodes can see the moving target simultaneously. In order to

determine the orientation $\theta_k$ of a sensor node $k$, the pin-hole model of a camera is used [1].

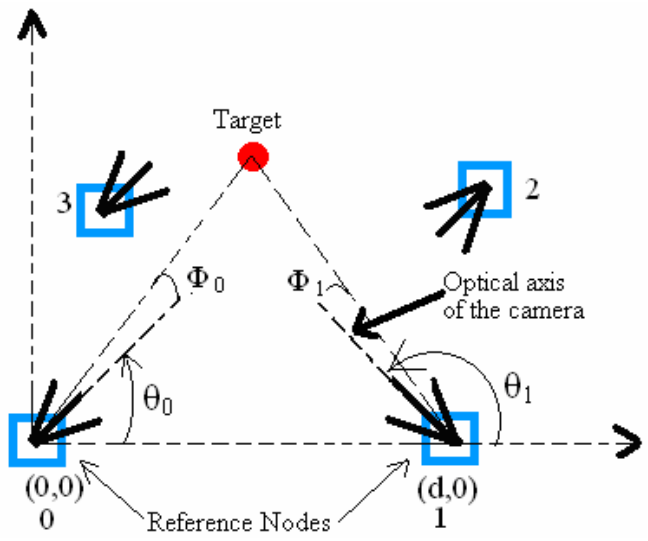



Figure 2.2: System model

The pin-hole model of a camera shown in Figure 2.3 helps a sensor node to calculate the angular offset $\phi$ between the target and its camera [1].

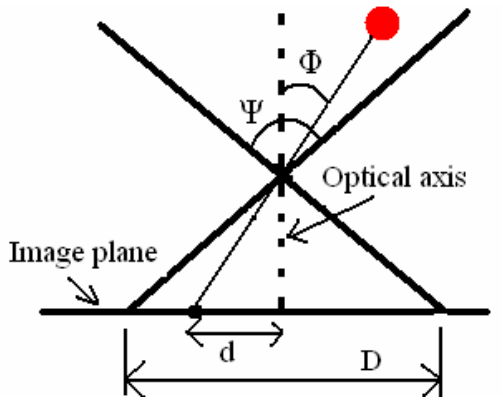



Figure 2.3: Pin-hole camera model [1]

From the Figure 2.3, the angular offset between the camera of a sensor node and the target in its field of view is given as [1]

$$\phi = \tan^{-1}\left(\frac{2d}{D} \times \tan(\frac{\psi}{2})\right)$$

D is the horizontal resolution of the camera in pixels, d is the distance in pixels between center of the image plane and the point on the image plane where the image of the target is formed and Ψ is the angular span of the field of view of the camera.

## 2.5 Blob detection

The camera records its images in frames from time to time. Initially, every camera in the network stores a frame which contains the environment that it can see without the presence of any moving object in its field of view. This frame is the static background frame of the camera and is updated regularly from time to time. Whenever a moving target enters the field of view of a camera, the camera of that sensor node captures a new frame and uses a frame differencing method and filters out the static background. This helps the camera detect the moving objects. The difference between the current frame and the static background frame help the camera identify the blob of the target. As the target moves, frame differencing is employed on the successive frames captured by the camera. The change in the blob positions from one frame to another is used to determine the horizontal shift in the center of the target's image in the image plane of the camera.

## 2.6 Distributed calibration algorithms

### 2.6.1 Calibrating the reference nodes and localizing the target

In Figure 2.4, reference node 0 is at the origin of a reference frame. Reference node 1in Figure 2.4 is at a unit distance from node 0. Let these two nodes be oriented at unknown angles $\theta_0$ and $\theta_1$ with respect to the horizontal respectively as shown in Figure 2.4. The

moving target is simultaneously tracked by both the reference nodes. Let $\phi_k^n$ be the angular offset between the camera of a sensor node $k$ and the moving target at the $n^{th}$ observation. The reference nodes extract the center of the blob of the target and use the pin-hole camera model to determine the angular offsets $\phi_{0,}^0 \phi_1^0$ respectively. Let us assume that the target has moved from location 1 to location 2 (as shown in Figure 2.4) while maintaining itself in the field of view of the reference nodes. When the object moves to location 2, let the distance and the heading recorded by the position sensor be denoted by the vector $Pe^{j\theta}$.

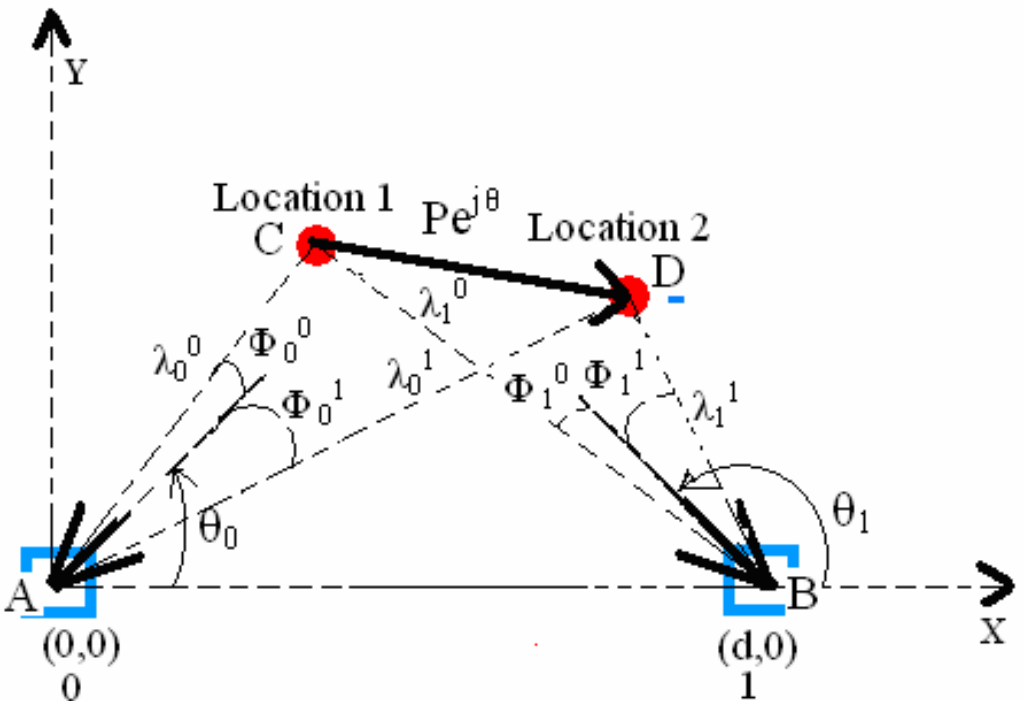



Figure 2.4: Target moved from location 1 to location 2

The reference nodes firstly determine the angular offsets $\phi_{0,}^1 \phi_1^1$ for their detected blob readings using the pin-hole camera model. These values are exchanged over the wireless network between the reference nodes. For every $n^{th}$ observation of the target made by

the sensor node, there is an unknown distance $\lambda_k^n$ between the sensor node $k$ and the target. So for the two different target positions in the field of view of the nodes, we have $\lambda_0^0$ , $\lambda_0^1$ for reference node 0 and $\lambda_1^0$ , $\lambda_1^1$ for reference node 1 respectively.

Applying the sine rule on the triangle ABC in Figure 2.4, the following relation can be obtained

$$\frac{\lambda_1^0}{\sin(\theta_0+\phi_0^0)} = \frac{\lambda_0^0}{\sin(\theta_1+\phi_1^0)} = \frac{d}{\sin[(\theta_1+\phi_1^0)-(\theta_0+\phi_0^0)]} \tag{1}$$

Applying the sine rule on the triangle ADB in Figure 2.4, the following relation can be obtained

$$\frac{\lambda_1^1}{\sin(\theta_0-\phi_0^1)} = \frac{\lambda_0^1}{\sin(\theta_1-\phi_1^1)} = \frac{d}{\sin[(\theta_1-\phi_1^1)-(\theta_0-\phi_0^1)]} \tag{2}$$

From the triangles ACD and BDC of Figure 2.4, equations (3) and (4) are obtained.

$$\lambda_0^1 e^{j(\theta_0-\phi_0^1)} = Pe^{j\theta} + \lambda_0^0 e^{j(\theta_0+\phi_0^0)} \tag{3}$$

$$\lambda_1^0 e^{j(\theta_1+\phi_1^0)} = -Pe^{j\theta} + \lambda_1^1 e^{j(\theta_1-\phi_1^1)} \tag{4}$$

Using (1), (2), (3) and (4), as set of 6 equations can be obtained which are shown below.

$$r_1 = (\lambda_1^0 \times \sin[(\theta_1+\phi_1^0)-(\theta_0+\phi_0^0)]) - (d \times \sin(\theta_0+\phi_0^0)$$
$$r_2 = (\lambda_0^1 \times \sin[(\theta_1+\phi_1^0)-(\theta_0+\phi_0^0)]) - (d \times \sin(\theta_1+\phi_1^0)$$
$$r_3 = (\lambda_0^1 \times \sin[(\theta_1-\phi_1^1)-(\theta_0-\phi_0^1)]) - (d \times \sin(\theta_1-\phi_1^1)$$
$$r_4 = (\lambda_1^1 \times \sin[(\theta_1-\phi_1^1)-(\theta_0-\phi_0^1)]) - (d \times \sin(\theta_0-\phi_0^1)$$
$$r_5 = (\lambda_0^1 \times \cos(\theta_0-\phi_0^1)) - (\lambda_1^1 \times \cos(\theta_1-\phi_0^1)) - (\lambda_0^0 \times \cos(\theta_0+\phi_0^0)) + (\lambda_1^0 \times \cos(\theta_1+\phi_1^0))$$
$$r_6 = (\lambda_0^1 \times \sin(\theta_0-\phi_0^1)) - (\lambda_1^1 \times \sin(\theta_1-\phi_0^1)) - (\lambda_0^0 \times \sin(\theta_0+\phi_0^0)) + (\lambda_1^0 \times \sin(\theta_1+\phi_1^0))$$

Then define a vector $\bar{r} = [r_1\, r_2\, r_3\, r_4\, r_5\, r_6]^T$. Define a vector of unknown parameters as $\bar{x} = [\theta_0\, \theta_1\, \lambda_0^1\, \lambda_0^2\, \lambda_1^1\, \lambda_1^2]^T$. We estimate the vector $\bar{x}$ using the Gauss-Newton method. The Jacobian matrix of the Gauss-Newton method is given by

$$J = \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \frac{\partial r_1}{\partial x_3} & \frac{\partial r_1}{\partial x_4} & \frac{\partial r_1}{\partial x_5} & \frac{\partial r_1}{\partial x_6} \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \frac{\partial r_2}{\partial x_3} & \frac{\partial r_2}{\partial x_4} & \frac{\partial r_2}{\partial x_5} & \frac{\partial r_2}{\partial x_6} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \frac{\partial r_6}{\partial x_1} & \frac{\partial r_6}{\partial x_2} & \frac{\partial r_6}{\partial x_3} & \frac{\partial r_6}{\partial x_4} & \frac{\partial r_6}{\partial x_5} & \frac{\partial r_6}{\partial x_6} \\ \end{bmatrix}$$

The update rule for Gauss-Newton method is given as follows [1]

$$\bar{x}_{i+1} = \bar{x}_i - [(J^T(\bar{x}_i) \times J(\bar{x}_i))^{-1} \times (J^T(\bar{x}_i) \times \bar{r}(\bar{x}_i)] \tag{5}$$

Steepest descent method [1] is used to reduce complexity and equation (5) can be simplified as

$$\bar{x}_{i+1} = \bar{x}_i - [\alpha^i J^T(\bar{x}_i) \times \bar{r}(\bar{x}_i)] \tag{6}$$

Here $\alpha^i$ is a positive step size.

The Gauss-Newton method is used to determine the orientations of the two reference nodes and also determine the moving target's coordinates at positions 0 and 1. Once the reference nodes have been calibrated, node 0 transmits the current location of the moving target back to the target. From here on as the target moves, it updates its current position in the reference coordinate system by applying dead reckoning on the readings it obtains from its position tracking sensor.

2.6.2 Calibrating the remaining nodes

Assuming that the moving target has been detected by an uncalibrated sensor node ($n$), our aim is to find the coordinates $(x, y)$ of this node $n$ and also determine the angle of orientation $\theta$ of its camera with respect to the positive x axis of the reference coordinate system as shown in the Figure 2.5 below.

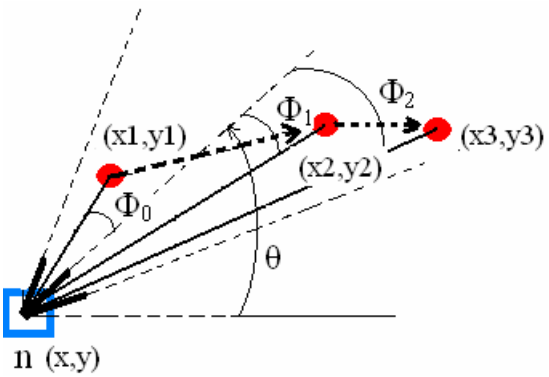



Figure 2.5: Target at three distinct locations in the field of view of an uncalibrated camera sensor node.

Let us assume the target has entered the field of view of an uncalibrated camera of a sensor node $n$. The camera sensor node requires the target to stop at three distinct locations (as shown in the Figure 2.5) in its field of view in order to calibrate itself. The camera of the sensor node extracts the coordinates of blob center of the target at the three distinct locations in its field of view. While extracting the blob coordinates, the node also requests the target to transmit its current location coordinates using the wireless connectivity. The sensor node then uses the pin-hole model to determine the angular offsets $\phi_0, \phi_1, \phi_2$ at target positions 0, 1 and 2 respectively.

Let $(x_1, y_1), (x_2, y_2)$ and $(x_3, y_3)$ be the coordinates of the moving target at positions 1, 2 and 3 respectively as shown in Figure 2.5.

Using the expression to find the slope of a straight line joining $(x\ , y\ )$ and $(x_1, y_1)$ , the equations (7), (8) and (9) can be obtained.

$$\tan(\theta + \phi_0) = \frac{(y_1 - y)}{(x_1 - x)} \tag{7}$$

$$\tan(\theta - \phi_1) = \frac{(y_2 - y)}{(x_2 - x)} \tag{8}$$

$$\tan(\theta - \phi_2) = \frac{(y_3 - y)}{(x_3 - x)} \tag{9}$$

In the above three equations, there are three unknowns $x$ , $y$ and $\theta$. MATLAB symbolic toolbox functions 'solve' and 'simplify' were used to obtain the solutions for the equations (7), (8) and (9) and a unique solution was obtained. In the above procedure, MATLAB gives the value for solution for $\theta$ only in the range [-90˚, 90˚]. However, to calculate the final angle in the range [0˚, 360˚] with respect to the positive X axis the following procedure is used. The orientation angle will be closest to that value of $\phi$ which is the least of the angles $\phi_0$ , $\phi_1$ and $\phi_2$ .

Select coordinates $(x_i, y_i)$ among $(x_1, y_1), (x_2, y_2)$ and $(x_3, y_3)$ for which the corresponding absolute $\phi$ value is the least. Assume that the origin of a coordinate system lies at the calculated coordinates $(x, y)$ of the camera. Determine the quadrant in which the coordinates of the point corresponding to the least value of $\phi$ would lie. This scenario is shown in the Figure 2.6.

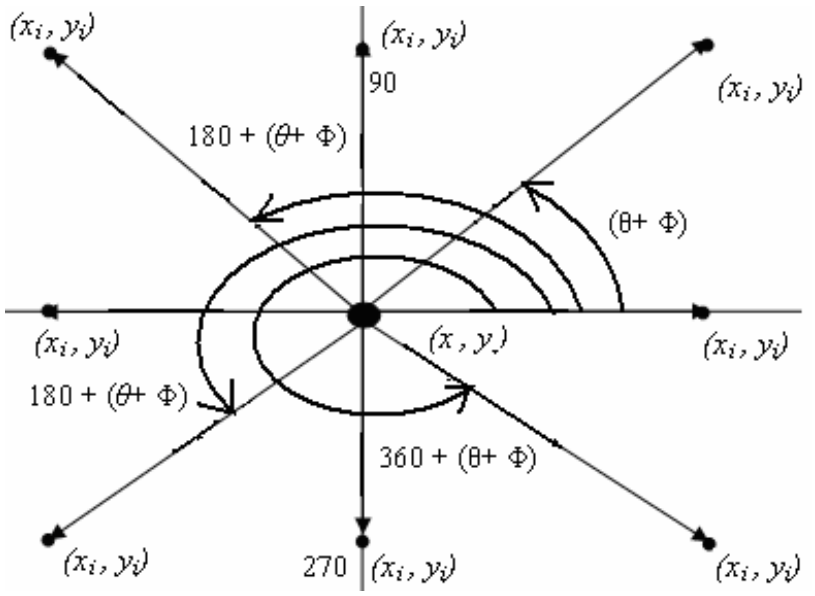



Figure 2.6: Possible Scenarios for determining the orientation of the camera

Depending on the quadrant in which the point $(x_i, y_i)$ lies with respect to this origin, use one of the below expressions to determine the final orientation of the camera.

Final Orientation = $\theta$ if $(x_i > x)$ and $(y_i >= y)$

Final Orientation = $360 + \theta$ if $(x_i > x)$ and $(y_i < y)$

Final Orientation = 90 if $(x_i = x)$ and $(y_i > y)$

Final Orientation = 270 if $(x_i = x)$ and $(y_i < y)$

Final Orientation = $180 + \theta$ if $(x_i < x)$ and any $y_i$

Thus, the final orientation of the camera, with respect to the positive X axis of reference coordinate system, is calculated depending on the angle information obtained by solving the equations (7), (8) and (9).

The expressions for $x, y$ and $\theta$ obtained from MATLAB and the final orientation calculation procedure, involve very simple arithmetic calculations. As such this calibration algorithm can be implemented on sensor nodes having a processor with

sufficient memory and which can perform simple floating point arithmetic calculations. Thus, by using this method, an uncalibrated camera sensor node is able to determine its coordinates as well as orientation in the reference coordinate system. After performing the calibration, the node also transmits this data back to the target. Thus, the moving target stores the location and orientation information from all the nodes it encounters in its path.

# CHAPTER 3

# IMPLEMENTATION OF THE CALIBRATION ALGORITHMS ON A THREE-NODE CAMERA SENSOR NETWORK

The calibration algorithm proposed in section 2.6 was practically implemented on a network of three sensor nodes. We present a brief description of the hardware test-bed built for this purpose. In this chapter, we describe the bottlenecks encountered while conducting the real time experiments and also provide alternatives to overcome the bottlenecks during the experimental implementation. The experiment results followed by a small discussion are presented in section 3.4.

## 3.1 Camera sensor module

The camera sensor node used for the experimental implementation of the calibration algorithm is shown in Figure 3.1.

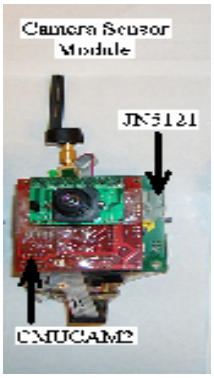



Figure 3.1: Camera sensor node comprising CMUCAM2 and JN5121

Each sensor node consists of a CMUCAM2 CMOS camera interfaced to the JN5121 wireless module from Jennic Corporation [3]. The Jennic module consists of a JN5121 wireless microcontroller which has an inbuilt programmable IEEE 802.15.4 protocol stack [23] [36].

## 3.2 CMUCAM2 camera

### 3.2.1 Description of the camera

The CMUCAM2 [2] camera module consists of the SX52 microcontroller interfaced to the OV6620 [26] or OV7620 omni vision CMOS camera, having a field of view of 45 degrees. CMUCAM2 vision sensor has the ability to track user defined color blobs at up to 50 frames per second. It has an adjustable image resolution of up to 166 x 255 pixels [2]. The CMUMCA2 camera can be interfaced to an external device using either a RS-232 communication link or TTL serial port. The CMUCAM2 is factory programmed with image processing algorithms to track a moving target, capture the image, determine coordinates of the centroid of the target's detected blob and calculate the mean and variance for the detected blob. The CMUCAM2 is programmed to respond to a set of predefined commands, sent by external devices over its serial port receive pin, to perform a set of operations. These commands are ASCII characters, using which interfacing devices can instruct the module to transmit pixel wise image values of a captured image frame, transmit the coordinates of the centroid, and calculate and transmit the mean and variance of the detected blob.

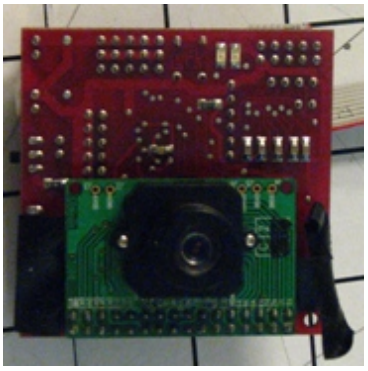


Figure 3.2: The CMUCAM2 vision sensor.

### 3.2.2 Color tracking by the CMUCAM2 camera

Color tracking is the ability to isolate a particular color from a captured color image and determine the coordinates of the region in the image that contains the color [2]. In order to track an object, the tracking device must have the knowledge of the color of the object. Any color can be represented by a combination of the three basic colors red, blue and green in proper proportions. Also, the surrounding light and the color of an object are not perfectly uniform at all the times and hence there is a need to accommodate these variations while performing color tracking. So, to track an object using CMUCAM2, it is necessary to specify the maximum and minimum allowable values for the red, green and blue channels within which the object color is likely to fall. Each channel color has a minimum value of 16 and a maximum value of 240. Once the channels bounds are set, the CMUCAM2 uses these bounds to process its captured images. For every image frame captured by its camera, the CMUCAM2 starts at the top left corner of the image and sequentially inspects every pixel of the image row by row. If a pixel being inspected inspecting falls in the range of colors that have been specified, it is marked as tracked. For a pixel that currently being tracked, CMUMCAM2 determines if the pixel's position

to check if it is at the top most, bottom most, left most or right most of all the other pixels that have already been tracked. If it finds that the pixel is outside of the current bounding box of the tracked region, it grows the bounding box so as to accommodate the current pixel. It uses this information to determine the centroid of the target's detected blob.

## 3.3 JN5121 wireless micro-controller module

The JN5121 is a low power and low cost IEEE 802.15.4 compliant wireless micro-controller [3]. This device combines an on chip 32-bit RISC core, a programmable IEEE 802.15.4 protocol stack operating at 2.4 GHz, 64KB of ROM and 96KB of RAM [3]. It operates in a 2.7 to 3.6 V DC voltage range. It has 21 digital Input - Output pins that enable the JN5121 to interface and control external devices like sensors and actuators. It has two programmable UART's (Universal Asynchronous Receiver and Transmitter) which help the JN5121 enable serial communication with external devices using the TTL level serial communication protocol.

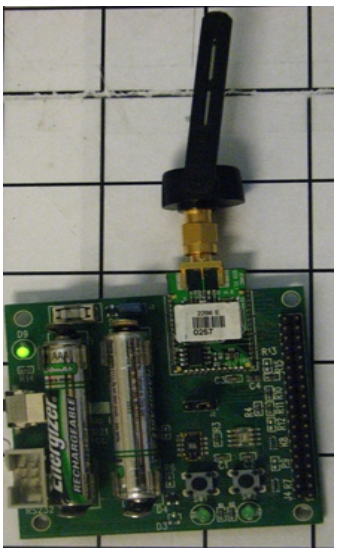



Figure 3.3: The JN5121 wireless micro-controller module.

## 3.4 Experiment Setup

### 3.4.1 Interfacing the CMUCAM2 to the JN5121

The CMUCAM2 module is interfaced to the JN5121 controller module using the wiring configuration shown in Figure 3.4. As shown in the Figure 3.4, the RS-232 serial port of CMUCAM2 vision sensor module can be interfaced with the UART0 of the JN5121 microcontroller. The operating voltage range for the RS-232 communication link is in the range of -12 to +12 volts. However, all the pins of the JN5121 work in the voltage range of 0 to 3.3 volts. So, an external level shifter module is used to level shift the signals between the RS-232 voltage logic and 3.3 volts logic on transmit and receive lines of the serial communication link as shown in the Figure 3.4. The JN5121 communicates with the CMUCAM2 at a baud rate of 38,400.

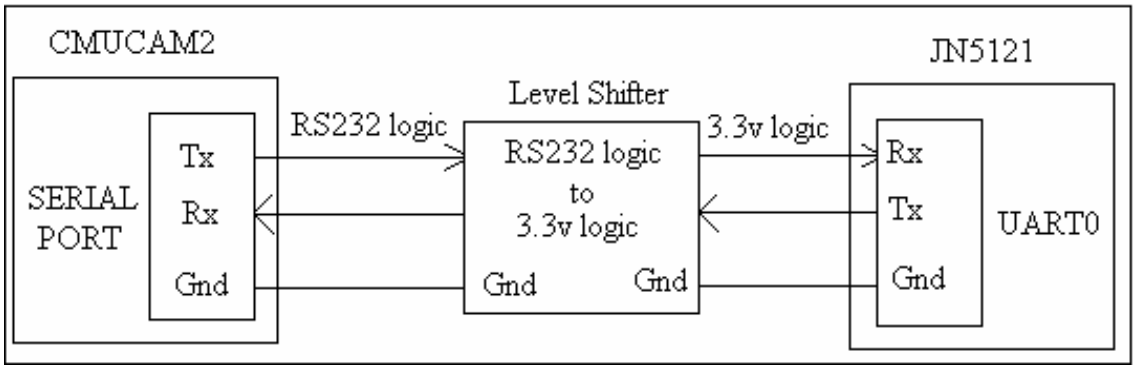



Figure 3.4: Interfacing the CMUCAM2 to the JN5121 controller.

### 3.4.2 Hardware setup

The Figure 3.5 shows the test bed used for the experiment. Four JN5121 modules were used to form a wireless sensor network. Of them, one node, called the 'Coordinator', is

programmed to start the zigbee network [23] and allows other devices to join the network. The remaining nodes, called as the 'End-Devices' are programmed to join the network started by the coordinator.

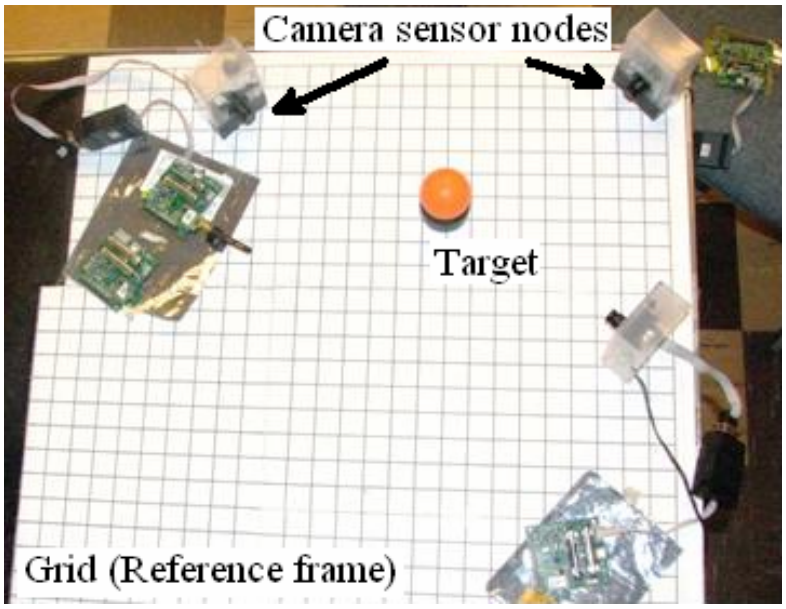



Figure 3.5: Test bed to implement the calibration algorithm.

The end-devices are also programmed to communicate with the CMUCAM2 cameras. Of the three end-device nodes, two nodes are selected as reference nodes. The origin of the reference coordinate system is taken to be present at one node while the other reference node was placed at 10 units from the node at the origin. The units in the X and Y direction are in inches. The third camera sensor node is placed at a random location with a random orientation. An 'orange' colored ball is taken as the moving target for the purpose of calibration. However, there are some bottlenecks encountered during the implementation of the experiment in accordance with the proposed algorithm.

### 3.4.3 Bottlenecks in the experiment

The distributed calibration algorithm requires each node to perform mathematical operations on its own and implement the Gauss-Newton estimation algorithm on the exchanged data values (for the reference nodes). The JN5121 has very limited computational abilities and cannot support floating point division and multiplication operations as required by the calibration algorithm. However, processors running on AVR and ARM architectures like the ATmega16 and LPC2106 can support floating point arithmetic operations. Due to the availability of Jennic processors, the calibration algorithm is run in a centralized manner with all the required calculations being performed on a PC. All the nodes send their camera readings of the moving target to the coordinator of the network. The coordinator is connected to a PC. The PC runs a VB.NET application which reads the information from the coordinator over the serial port and stores the data pertaining to all the three cameras to text files. In MATLAB, a program runs the calibration algorithm on the data readings saved in the above mentioned files.

Another limitation of this experiment is that the orange ball is a passive object and cannot communicate with the wireless sensor nodes as required by the calibration algorithm. So in order to run the calibration experiment and imitate a cooperative friendly target, the various locations at which the target has to be placed are pre determined. Also, the target used in this experiment is not equipped with any position sensor to calculate the distance and angle information values between various points in its path. In order to imitate a position tracking sensor, the distance and angle information values between the

various pre determined target locations are calculated and are used as inputs to the calibration algorithm.

### 3.4.4 Software

On joining the network, each end-device of the network is assigned an 8 bit distinct ID by the coordinator. The coordinator uses this ID to send commands to a specific end-device. VB.NET and MATLAB were used for the purpose of implementing the calibration on the test bed shown in Figure 3.5. The VB.NET application uses the coordinator to send commands to control the cameras of the end-devices. The VB.NET application uses the RS-232 Class to talk to the coordinator node of the zigbee network using a serial cable. On receiving a message from the PC, the coordinator checks for the end-device address to which the packet is addressed. It then transmits this data packet to that specific end-device.

On receiving the data packet, the end-device decodes the control command embedded in the packet and forwards the control command via the RS-232 link. On receiving the command, the CMUCAM2 performs the required operations and returns the results to the end-device. The end device then builds a data packet of the response data and transmits it to the PC via the coordinator. The VB.NET application running on the PC decodes the received data, identifies the transmitting end-device and stores the data from each end device to a separate file. In order to perform the calibration algorithm, the blob coordinates of the tracking ball from each camera is requested by the PC. Whenever the tracking ball is placed at a location in the test bed, the VB.NET application is used to request packets containing the centroid information only from those end-device cameras,

which contain the tracking ball in their field of view. Likewise the tracking experiment is repeated by placing the ball at each predetermined location. Each end device is allocated a separate file by the PC to store its corresponding track values. The data from these files along with the position vectors between each target location are fed to a MATLAB program. This program firstly uses the Gauss-Newton method to localize the target ball and uses the position vector readings to build the target path. Simultaneously, it calibrates the position and orientation for each reference node. A MATLAB program, by using track values and the corresponding target positions associated with each track value, runs the calibration algorithm and determines the position coordinates and orientation of the remaining camera sensor nodes. A flow chart for the experimental procedure is shown in the Figure 3.6

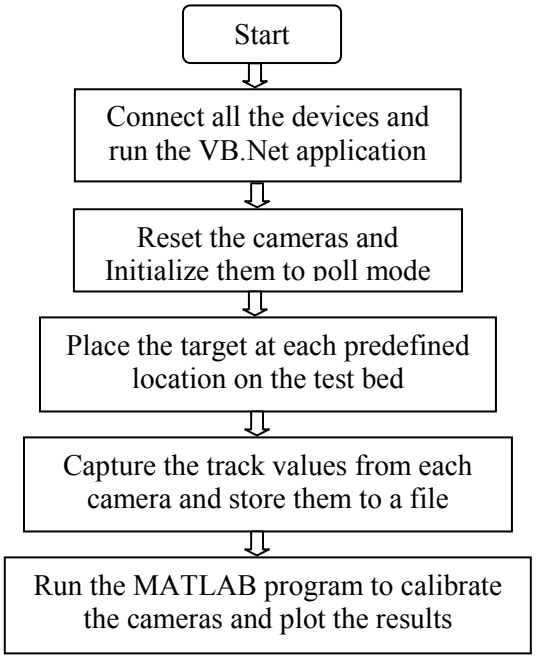



Figure 3.6: Flow-Chart for the experiment.

## 3.5 Experimental Results

Experiments were carried out on the test bed shown in Figure 3.5. The two reference nodes were fixed at (0, 0, 45˚) and (14, 0,135˚). The calibration algorithm was run for various positions and orientations of Camera 3 and the results are shown below.

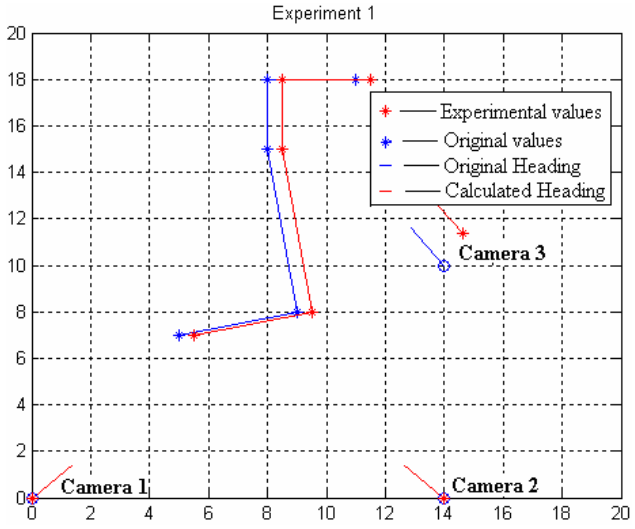



Figure 3.7: Result plot for experiment 1.

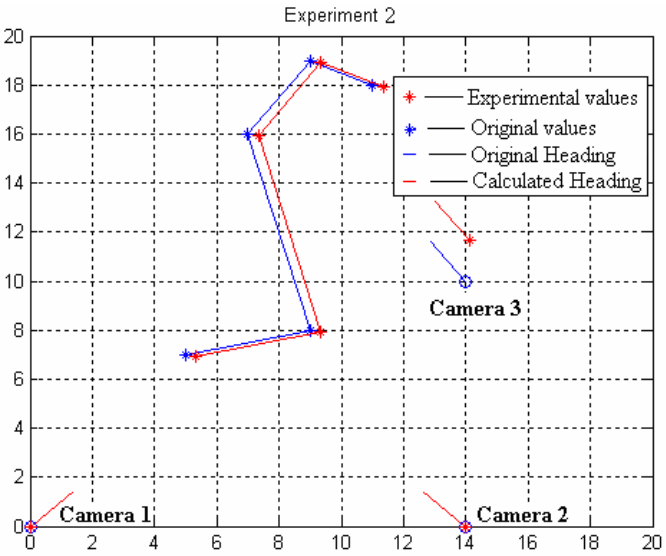



Figure 3.8: Result plot for experiment 2.

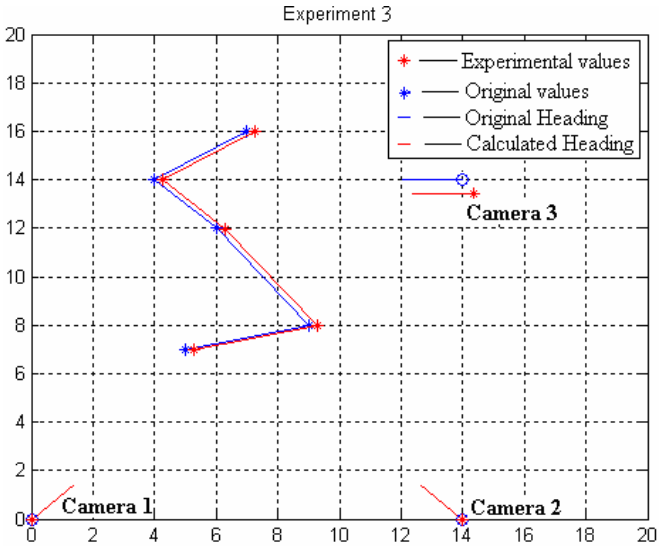



Figure 3.9: Result plot for experiment 3.

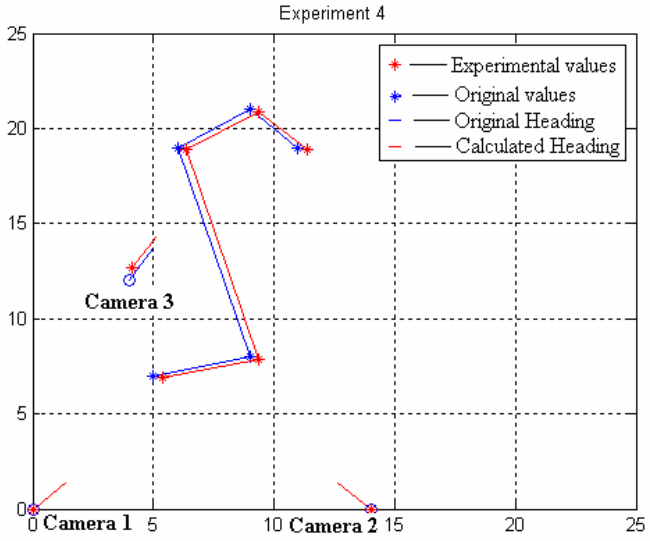



Figure 3.10: Result plot for experiment 4.

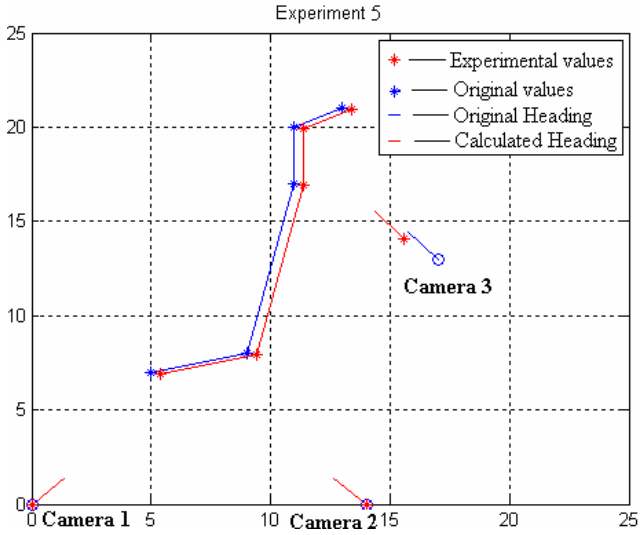



Figure 3.11: Result plot for experiment 5.

The results from the five experiments were statistically analyzed. The table below shows the mean, variance and standard deviation in the X, Y and θ values for camera 3 for five experiments.

| Statistical parameter | X value (inches) | Y value (inches) | θ value (degrees) |
|---|---|---|---|
| Mean Error | -0.0220 | 0.9360 | -1.8400 |
| Standard Deviation | 0.796 | 0.8736 | 7.1618 |

Table 3.1: Statistical parameters for the pose of camera 3 for five experiments

# CHAPTER 4

# SIMULATION FOR A LARGE SCALE DISTRIBUTED CAMERA SENSOR NETWORK

Due to the limited availability of resources, the calibration algorithm proposed in section 2.6 was practically implemented on a network of the three camera sensor nodes only. In order to test its reliability and scalability with large networks, the proposed calibration algorithm was simulated for a large scale wireless sensor network using the Player/Stage robotic simulation software [4], [11]. In this chapter we present an overview of the Player/Stage software. We use this software to simulate the proposed calibration algorithm on a distributed network of 13 camera sensor nodes.

## 4.1 Player/Stage robotic simulation software

There is always a growing need to develop and test algorithms for complex and large scale heterogeneous environments and applications. Such an extensive research is always accompanied by the limited availability of resources like time and money. This has led researchers look for middleware platforms and tools that can help them develop and test context-aware applications using simulations [4]. Player/Stage is one such open source middleware platform. It is an open source software available for free. It is widely for

research in robotics and sensor systems [5]. Player/Stage is an integration of two platforms. The first platform is the player robot device server and the second platform is the stage which is a 2-dimensional simulator [6]. It was developed at the University of Southern California's Robotic research lab and is currently hosted on sourceforge.net. Below is a block diagram for Player/Stage.

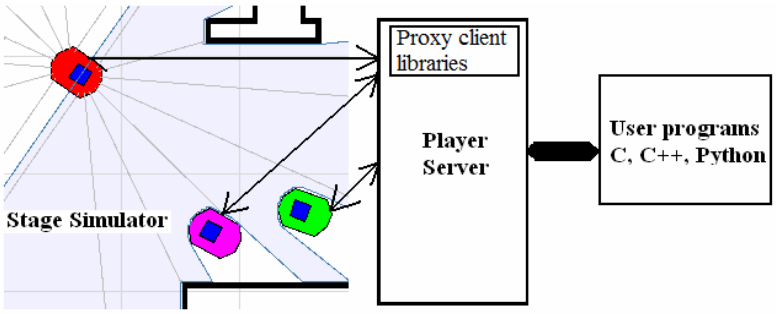



Figure 4.1: Block diagram of Player/Stage robotic simulation software

### 4.1.1 Player

The Player is a robot device server and contains libraries for various robots, sensors and actuators [6], [4]. It can be used to program and control real time robots and sensor using a TCP/IP connection [7]. On a computer with network connectivity, Player can be used with various programming languages like C, C++ to write programs to control the real time devices. Every device is made up of an interface and a driver. An interface provides a specification to interact with a certain class of robotic sensor, actuator, or an algorithm [9]. It defines the syntax and semantics of all messages or data that can be exchanged between the entities of the same class [9]. A driver is software that talks to device or algorithm, and translates its inputs and outputs in the format specified by the device interface [9]. Though it can be used to control hardware, Player also contains several

useful virtual devices programmed with sensor pre-processing or sensor-integration algorithms that be used to build powerful robot controllers.

### 4.1.2 Stage

Stage is a simulator which can be used to simulate a multi agent autonomous system in 2-dimesional bitmapped environments. Stage provides virtual Player robots which interact with simulated devices rather than physical ones. It provides a virtual world that can be used to simulate real time scenarios. The world can be populated by mobile robots and sensors, along with the various objects which robots can sense. To emulate a real world device in a simple and a computationally less intensive way, the stage provides a model file. Stage provides models to emulate a variety of sensors and actuators like sonar or infrared rangers, scanning laser rangefinder, color-blob tracking, fiducial tracking, bumpers, grippers and mobile robot bases with odometric or global localization [10]. Stage devices present a standard Player interface so few or no changes are required to move between simulation and hardware. The devices are accessed through Player, as if they were real hardware. Stage aims to mimic real world devices more efficiently rather than accurately [11].

## 4.2 Simulation setup

### 4.2.1 Description of the environment used for the simulation

In the stage part of the simulation experiment, the world file defines the environment setting for the experiment. The world is of size 30 x 30. It consists of 13 camera sensor

nodes numbered 1 to 13 and a tracking object. The numbers 1 to 13 are also the unique fiducial id's of respective nodes. Each camera sensor node is equipped with a *blob finder* interface, a *fiducial finder* interface and an *opaque* interface. The tracking target is equipped with a *position2d* interface, a *fiducial finde*r interface and an *opaque interface*. The tracking target has a fiducial id of 20. Of the 13 camera sensor nodes, nodes numbered 1 and 2 are used as reference nodes. These nodes are placed at locations (0, 0) and (10, 0) respectively but at unknown orientations. These two nodes form the reference coordinate system. The remaining 11 nodes are placed at unknown locations and orientations in the coordinate system. The world used for the simulation experiment is shown below in the Figure 4.2.

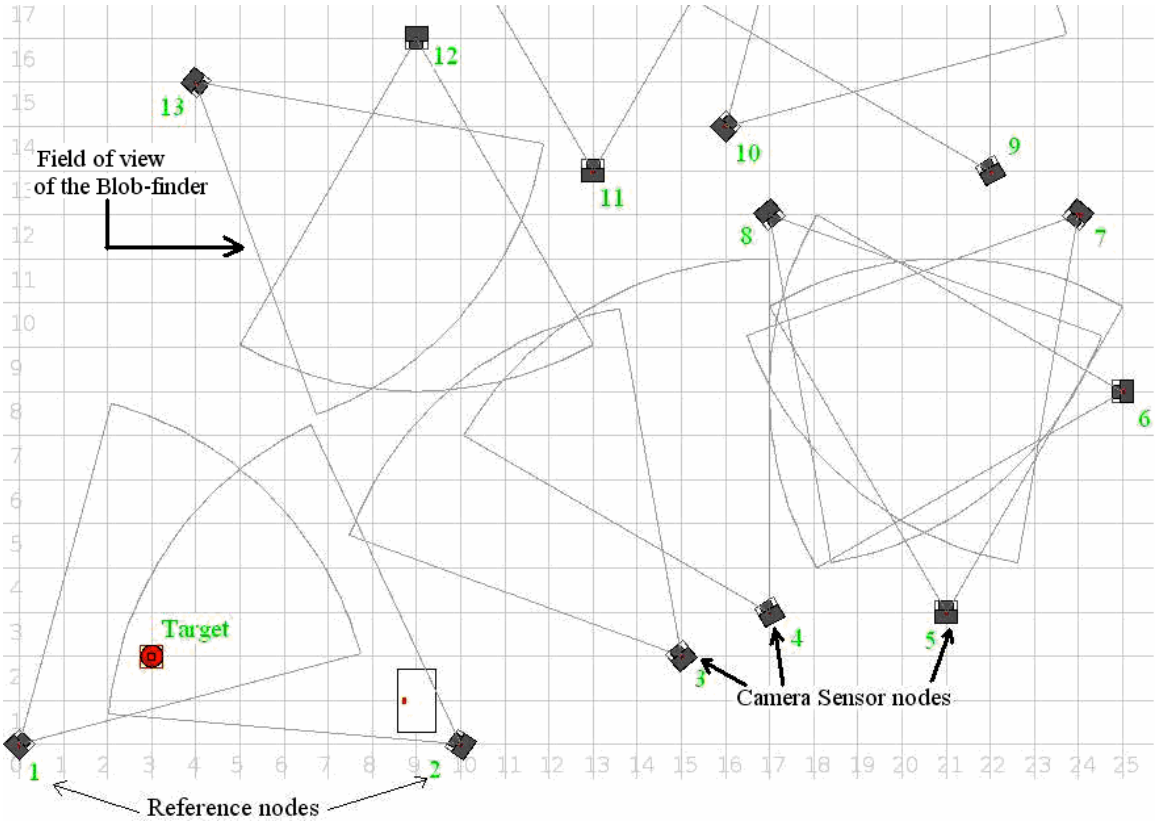



Figure 4.2: Stage world used for the simulation

### 4.2.2 The tracking target and its interfaces

The tracking target is used to calibrate the camera sensor nodes in the environment and collect their location and orientation information. The tracking target is an orange red colored mobile robot equipped with a *position2d* interface, a *fiducial finder* interface and an *opaque interface*. The *position2d* interface consists of motors that add mobility to the robot. The motion of the robot can be controlled by passing appropriate steering commands to these motors. In order to calibrate a camera sensor node, the tracking target has to identify the presence of the sensor node in its nearby environment. As there is no wireless communication interface provided by the Player/Stage, the tracking target makes use of the *fiducial finder*. Whenever a camera sensor node falls in the fiducial range of the moving target, the target stores the fiducial ID of the detected camera sensor node. This helps the target decide which node to communicate with. In order to share its current location information with the nearby camera sensor node, the tracking target is equipped with an *opaque* interface which sets up a communication channel between a sensor node and the tracking target.

### 4.2.3 The camera sensor node and its interfaces

In Figure 4.1, the gray colored objects in the world are the camera sensor nodes. Each camera sensor node is equipped with a *blob-finder*, a *fiducial finder* and an *opaque* interface. In order to calibrate its location and orientation, the camera sensor node should be able to detect the tracking target. The *blob finder* interface has a field of view of 60˚ and a range of 8 units. The color channel of the *blob finder* is configured to detect only

the “OrangeRed” color which is the color of the tracking object used in the experiment. Whenever the target enters its field of view, the *blob finder* interface of the sensor node takes continuous readings of the target. This however leads to memory overflow issues due to the limited queue size allotted to the blob finder model. This overflow issue can be resolved by programming the sensor node to take only one blob finder reading by stopping the target at a location in the field of view of the camera. In order to determine whether the target has stopped in its range, the sensor node is equipped with a *fiducial finder* interface. The *fiducial finder* returns the X and Y coordinates of the detected fiducial target with respect to the detecting sensor node. Once a target enters the fiducial range of the sensor node, the sensor node scans for the changes in the X and Y coordinates between its successive fiducial readings. If the target has stopped moving, the change in the above coordinates for two successive fiducial readings will be ‘0’. The sensor node now uses its *blob finder* interface to determine the presence of the target blob in its field of view. Once a stationary target blob has been detected by the sensor node, the sensor node uses its *opaque* interface to request the current position coordinates from the tracking target. Once calibrated, the sensor node uses the same communication channel of the *opaque* interface to send back its position and orientation information back to the target.

## 4.3 Simulation

The simulation is divided into two phases. In the first phase, the cooperative moving target appears at two locations in the field of view of the reference nodes. The reference nodes use their respective *blob finders* to extract the coordinates of the centroid of the

target's detected blob. C++ programs were written to implement the calculations involved in the pin-hole camera model and Gauss-Newton method. The moving target is now localized in the reference coordinate system formed by the reference nodes. The first reference node communicates the current location coordinates to the friendly target using the communication channel setup by the *relay* interface. The calibration algorithm requires the moving target to be equipped with a wearable sensor so as to regularly update its location coordinates. Since Player/Stage does not provide any library to simulate a position tracking sensor, the target path was predetermined and position vector between various points where the target has to stop are pre-calculated. The moving target is programmed to accept this position vector whenever it stops. At its current stopped location, it uses the position vector and computes its next location. It then steers itself to the next calculated location using closed loop control mechanism. The target path is selected in such a way that it stops at three distinct locations in the field of view of the blob finder of each of the sensor nodes except the reference nodes. Whenever the target stops in the field of view of the *blob finder* of an uncalibrated sensor node, it transmits its current location coordinates using the *relay (opaque)* interface connecting the target and the sensor node. Simultaneously the sensor node uses the *blob finder* readings to calculate the angle between the optical axis of *blob finder* and the line joining its *blob finder* center to the center of the target. Once the target has stopped at three locations in its field of view, the sensor node runs the calibration procedure to determine its position coordinates as well as its orientation with respect to the positive X axis. It also communicates the same information to the cooperative target. Likewise, the cooperative target while moving along its predefined path, calibrates the sensor nodes in the environment. It also

maintains a record of the locations and orientations of all the nodes it has encountered during its motion.

## 4.4 Simulation Results

The simulation was carried out on the world configuration shown in Figure 4.2 using three different target paths. The screenshots of the stage simulation are shown below in Figures 4.3, 4.4 and 4.5.

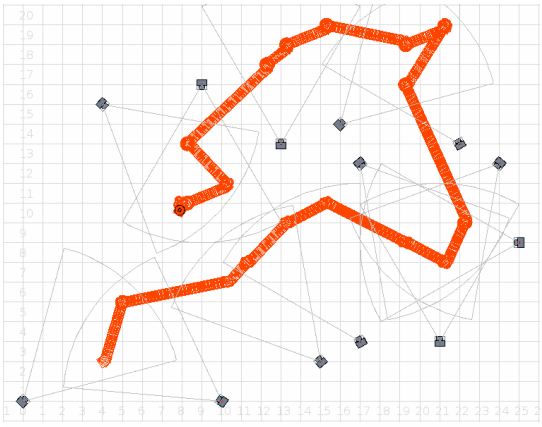


Figure 4.3: Path 1 trajectory in Stage

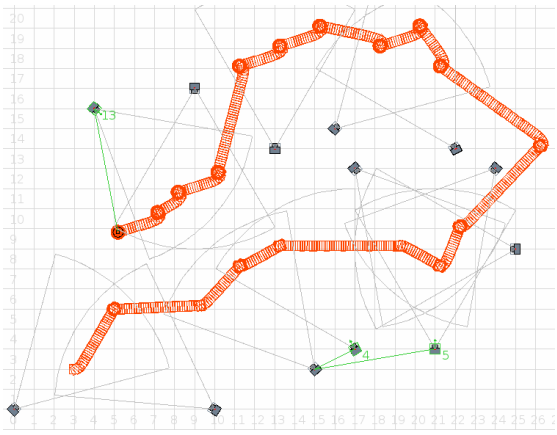


Figure 4.4: Path 2 trajectory in Stage

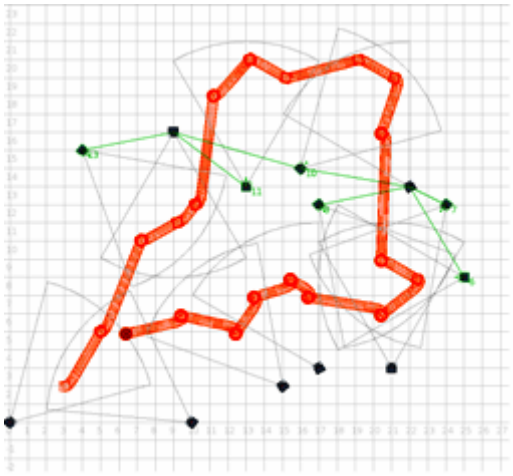


Figure 4.5: Path 3 trajectory in Stage

The coordinates and orientations of the sensor nodes obtained using the proposed calibration technique along the three target trajectories are plotted in Figures 4.6, 4.7 and 4.8 respectively.

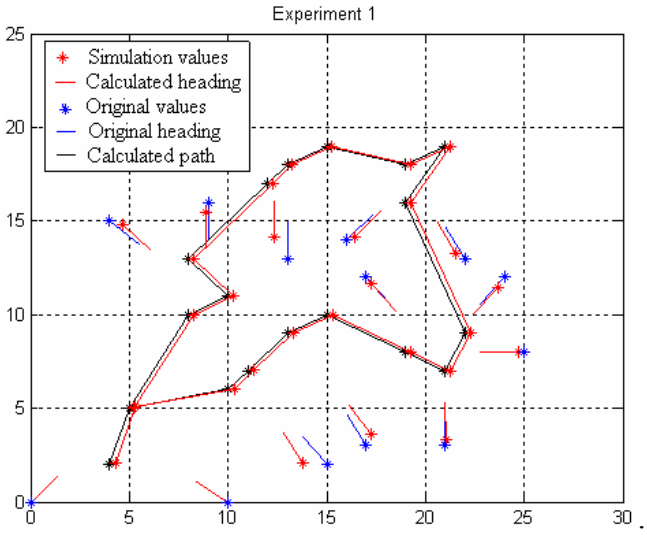



Figure 4.6: Plot for simulation on path 1

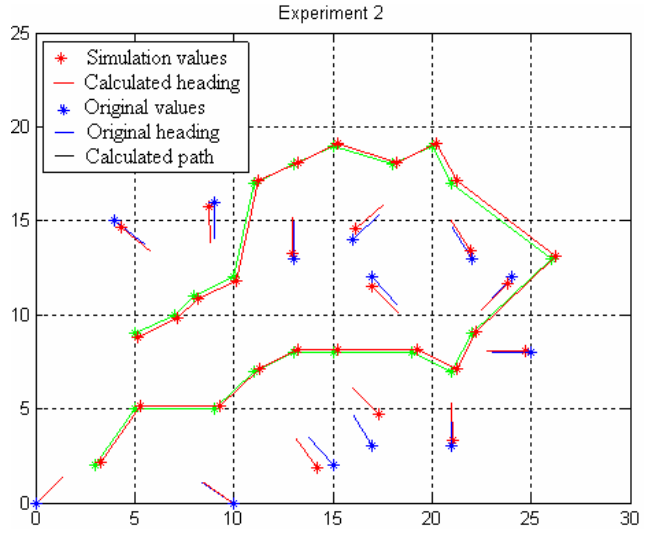



Figure 4.7: Plot for simulation on path 2

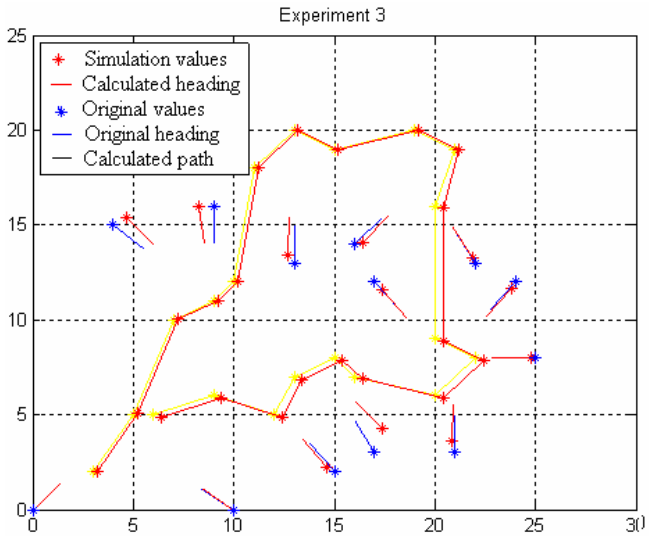



Figure 4.8: Plot for simulation on path 3

The table below shows the results of the sensor coordinates obtained by using the proposed calibration algorithm on each of the sensor nodes for target paths numbered 1, 2 and 3. The coordinates of a sensor nodes are represented as (X, Y, θ) in the below table. Here θ is in degrees.

| Sensor Node ID | Original Coordinates | Calculated Coordinates for path 1 | Calculated Coordinates for path 2 | Calculated Coordinates for path 3 |
|---|---|---|---|---|
| 1 | (0, 0, 45) | (0, 0, 45.39) | (0, 0, 45.83) | (0, 0, 44.66) |
| 2 | (10, 0 , 145) | (10.0, 0, 146.08) | (10.0, 0,144.42) | (10.00, 0, 144.8320) |
| 3 | (15, 2 , 130) | (13.81, 2.09, 121.58) | (14.22, 1.84, 124.15) | (14.64, 2.21, 128.67) |
| 4 | (17, 3, 120) | (17.22, 3.59, 124.45) | (17.32, 4.71, 133.50) | (17.37, 4.26, 132.60) |
| 5 | (21, 3, 90) | (21.05, 3.36, 91.13) | (21.02, 3.36, 91.36) | (20.84, 3.59, 88.11) |
| 6 | (25, 8, 180) | (24.70, 8.03, 180.69) | (24.70, 8.10, 181.67) | (24.75, 8.03, 181.30) |
| 7 | (24, 12, 230) | (23.70, 11.47, 228.91) | (23.79, 11.67, 228.50) | (23.79,11.65, 229.57) |
| 8 | (17, 12, 310) | (17.28, 11.67, 310.04) | (16.97, 11.51, 312.92) | (17.37,11.59, 309.60) |
| 9 | (22, 13, 120) | (21.50, 13.26, 116.80) | (21.95, 13.44, 122.30) | (21.90,13.27, 121.18) |
| 10 | (16, 14, 45) | (16.39, 14.18, 46.38) | (16.14, 14.56, 42.36) | (16.39, 14.10, 46.97) |
| 11 | (13, 13, 90) | (12.36, 14.14, 91.23) | (12.94, 13.27, 90.23) | (12.69, 13.43, 87.68) |
| 12 | (9, 16, 270) | (8.85, 15.47, 271.60) | (8.72, 15.77, 273.35) | (8.27, 15.97, 278.08) |
| 13 | (4, 15, 320) | (4.67, 14.78, 317.69) | (4.30, 14.64, 320.60) | (4.67, 15.36, 313.36) |

Table: 4.1: Table showing the original and calculated coordinates of the sensor nodes for three different target paths.

| Sensor Node ID | Error in Coordinates for path 1 | Error in Coordinates for path 2 | Error in Coordinates for path 3 |
|---|---|---|---|
| 1 | (0, 0, 0.39) | (0, 0, 0.83) | (0, 0, -0.33) |
| 2 | (0, 0, 1.08) | (0, 0, -0.58) | (0, 0, -0.16) |
| 3 | (-1.19, 0.09, -8.42) | (-0.78, -0.16, -5.85) | (-0.36, 0.21, -1.33) |
| 4 | (0.22, 0.59, 4.45) | (0.32, 1.71, 13.50) | (0.37, 1.26, 12.60) |
| 5 | (0.05, 0.36, 1.13) | (0.02, 0.36, 1.36) | (-0.16, 0.59, -1.89) |
| 6 | (-0.30, 0.03, 0.69) | (-0.30, 0.10, 1.67) | (-0.25, 0.03, 1.30) |
| 7 | (-0.30, -0.53, -1.09) | (-0.21, -0.33, -1.50) | (-0.21, -0.35, -0.43) |

| | | | |
|---|---|---|---|
| 8 | (0.28, -0.33, 0.04) | (-0.03, -0.49, 2.92) | (0.37, -0.41, -0.40) |
| 9 | (-0.50, 0.26, -3.20) | (-0.05, 0.44, 2.30) | (-0.10, 0.27, 1.18) |
| 10 | (0.39, 0.18, 1.38) | (0.14, 0.56, -2.64) | (0.39, 0.10, 1.97) |
| 11 | (-0.64, 1.14, 1.23) | (-0.06, 0.27, 0.23) | (-0.31, 0.43, -2.32) |
| 12 | (-0.15, -0.53, 1.60) | (-0.28, -0.23, 3.35) | (-0.73, -0.03, 8.08) |
| 13 | (0.67, -0.22, -2.31) | (0.30, -0.36, 0.60) | (0.67, 0.36, -6.64) |

Table: 4.2: Error in the coordinates of the sensor nodes for three different target paths.

The errors in the X, Y and θ for all the camera sensor nodes for three different target paths have been calculated and are shown in Table 4.2. Statistical parameters like mean and standard deviation are calculated on the sensor node coordinates obtained by the calibration algorithm. They are shown in the Tables 4.3, 4.4 and 4.5.

| Statistical parameter | X value | Y value | θ value |
|---|---|---|---|
| Mean Error | -0.1131 | 0.0800 | -0.2319 |
| Standard Deviation | 0.4867 | 0.4585 | 3.1069 |

Table 4.3: Statistical parameters for the coordinates of the cameras obtained on path 1

| Statistical parameter | X value | Y value | θ value |
|---|---|---|---|
| Mean Error | -0.0715 | 0.1438 | 1.2461 |
| Standard Deviation | 0.2849 | 0.5724 | 4.4407 |

Table 4.4: Statistical parameters for the coordinates of the cameras obtained on path 2

| Statistical parameter | X value | Y value | θ value |
|---|---|---|---|
| Mean Error | -0.0246 | 0.1892 | 0.8935 |
| Standard Deviation | 0.3838 | 0.4286 | 4.7952 |

Table 4.5: Statistical parameters for the coordinates of the cameras obtained on path 3

From the simulation results, it is clear that the proposed calibration algorithm, which was practically tested on a three node camera sensor network, can be extended to a large scale network.

# CHAPTER 5

# CONCLUSIONS AND FUTURE WORK

## 5.1 Conclusions

Recently, research in distributed camera calibration is gaining a lot of importance because of its ability to provide simple and precise solutions for monitoring and tracking applications. In this thesis we have presented some of the various calibration techniques proposed by many researchers. Many of these techniques require landmarks and complex image processing algorithms to calibrate a camera which may at times impose constraints on the power requirements of camera sensor nodes. Such techniques cannot be utilized to their full potential in real world scenarios. The calibration technique in [14], which does not require the use of landmarks, only gives the relative coordinates and orientation of a camera sensor node. Mapping this to a global frame using a mechanism like GPS is not only tedious but may also increase the power requirement of a sensor node. Another discussed calibration technique which localizes the target and calibrates the cameras in the same reference frame requires at least three or more camera sensor nodes to simultaneously track a target at five different locations. The self calibration protocol for a camera sensor network proposed in [15] requires four target observations to calibrate a camera. This calibration technique uses two different methods, one to find the coordinates of the camera and the

other to find its orientation. The work in the thesis tries to provide a novel solution for a distributed camera sensor network. We propose an algorithm which uses a cooperative target. It moves randomly in an environment, collaborates with the camera sensor nodes and assists them in performing distributed calibration. The calibration technique is simple and computationally less intensive. It combines the concept of pin-hole camera model proposed in [1] with the concept of the slope of a straight line to determine the location coordinates and the orientation of a camera sensor node. Unlike the other discussed methods that require any initial guesses for the external parameters of a camera, our calibration algorithm does not require any initial guesses and hence rules out the possibility of the getting stuck at a local minimum. We use MATLAB symbolic toolbox operations on to obtain the expressions for the coordinates and orientation of a camera. As the final expressions contain only simple arithmetic operations, they can be implemented on processors which can support floating point arithmetic. This calibration technique does not contain any iterative procedures and hence, will consume relatively less time and energy. This will also increase the life-span of the network. The proposed calibration algorithm was implemented practically on a three node camera sensor network. Laboratory experiments proved that the calibration algorithm resulted in a mean error of 0.022 inches in determining the X coordinate of the camera, a mean error of 0.93 inches in determining the Y coordinate and an error of 1.84˚ in determining the orientation of a camera. The reliability and scalability of this algorithm was tested by simulating it for a large scale camera sensor network using Player/Stage robotic simulation software. Simulation experiments have given a mean error of 0.0697 units, 0.1627 units and 0.7139˚ in calculating the $(x, y, \theta)$ values for a camera.

## 5.2 Scope for further development

The proposed calibration assumes that the cooperative target is equipped with a dead reckoning position tracking sensor. In the simulation, the target is moved along a predefined path using a closed loop control algorithm. As the target path is defined, the error in localizing the target is less. However, in a real world situation, the target moves randomly in an environment, once it is localized by the reference nodes. Thereafter, for every step it advances, the target uses dead reckoning to calculate its coordinates. With an increase in the number of target steps, the dead reckoning error would accumulate. This will lead to an error in the calculation of the coordinates by the target which in turn would lead to an error in calibrating a camera sensor node encountered by the target. There is a need to refine the target coordinates whenever the calibration of a camera sensor node is performed. This is called the simultaneous localization and tracking problem (SLAT). To solve for this problem, it is necessary to accurately model the motion of the target with appropriate parameters. Kalman filter [30] can be used to solve this problem. Kalman filter provides a precise recursive solution to estimate the state of a dynamic process which in the case is the location coordinates of the target as well as the location and orientation of the camera. An approach that deserves attention is modeling the target's motion and the external calibration parameters of the camera with appropriate state variables and use kalman filter to estimate and refine them. Most of the target localization techniques require at least two cameras to track a non-cooperative target if the coordinates and orientations of the cameras involved are known. In many situations, it may not be possible to find a region of overlap between two cameras and hence the target

localization is not possible. Hence there is a need to solve this problem only by using a single camera. In order to estimate the target coordinates with a single calibrated camera, the distance between the target and the camera is required. In [14], Kulkarni *et al.* suggest that with a known size of the object, it is possible to find the distance between the camera and the target. However, it is not possible to have prior information of the target size if the object is non-cooperative (for example an enemy soldier). So, we need to come up with a technique such that a camera can estimate the target size from its captured images. This information along with the optical parameters of the camera can be used to determine the distance between the target and the camera. The node can then mathematically combine the camera's external parameters like location and orientation with the distance information and determine the coordinates of the target's current location. The main advantage of this method would be that distributed target localization and tracking will be performed by using only a single camera. Also it will reduce the need to solve for the region of overlap between cameras, which will eliminate the need for image processing algorithms and exchange of image information between the cameras over the network.

REFERENCES

[1] Huang Lee, Hamid Aghajan, "Collaborative Node Localization using opportunistic Target Observations", *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks,* pages 9-18. October *2006*.

[2] CMUCAM2 data sheet at http://www.cs.cmu.edu/~cmucam2/. January 2007

[3] JN5121 wireless micro-controller datasheet at http://www.jennic.com/. January 2007

[4] Matthias Kranz, Radu Bogdan, Rusu2 Alexis Maldonado, Michael Beetz, Albrecht Schmidt, "A Player/Stage System for Context-Aware Intelligent Environments", *In Proceedings of UbiSys'06, System Support for Ubiquitous Computing Workshop, at the 8th Annual Conference on Ubiquitous Computing (Ubicomp,* September 2006*)*

[5] http://playerstage.sourceforge.net/index.php?src=index. October 2007

[6] http://playerstage.sourceforge.net/index.php?src=player. October 2007

[7] http://www-robotics.usc.edu/?l=Projects:PlayerStageGazebo. October 2007

[8] http://playerstage.sourceforge.net/doc/Player-2.0.0/player/. October 2007

[9] http://playerstage.sourceforge.net/doc/Player-2.0.0/player/group__tutorial__devices.html. October 2007

[10] http://playerstage.sourceforge.net/doc/Stage-2.0.0/group__stage.html. October 2007

[11] Matthias Kranz, Radu Bogdan, Rusu2 Alexis Maldonado, Michael Beetz, Albrecht Schmid, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor

Systems", In *Proceedings of the International Conference on Advanced Robotics* (ICAR *2003), Coimbra, Portugal,* pages 317-323. June 30 - July 3, 2003.

[12] http://playerstage.sourceforge.net/doc/Player-2.0.0/player/group__interfaces.html. October 2007

[13] Zhengyou Zhang, "A Flexible New Technique for Camera Calibration", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 22(11):1330–1334. NOVEMBER 2000.

[14] Purushottam Kulkarni, Prashant Shenoy, and Deepak Ganesan, "Approximate Initialization of Camera Sensor Networks", *In Proceedings of the Fourth European Conference on Wireless Sensor Networks (EWSN).* 2007.

[15] I.M. Rekletis, G. Dudek, "Automated calibration of a camera sensor network", *In IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton Alberta, Canada,* pp. 401–406. 2005,.

[16] M. Paskin S. Funiak, C. Guestrin and R. Sukthankar, "Distributed localization of networked cameras", In *ISPN*, 2006.

[17] Clarke, T.A. and Fryer, J.G. 1998, "The development of camera calibration methods and models", *Photogrammetric Record*, 16:51–66.

[18] Remondino F., Fraser C., 2006, "Digital camera calibration methods: considerations and comparisons", *InternationalArchives of Photogrammetry, Remote Sensing and the Spatial Sciences*, 36(5), pp. 266-272.

[19] A. Rowe, C. Rosenberg, I. Nourbakhsh, "A low cost embedded color vision system", *In Proceedings of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), Lausanne, Switzerland.* October 2002.

[20] X. Liu, P. Kulkarni, P. Shenoy, and D. Ganesan, "Snapshot: A Self-Calibration Protocol for Camera Sensor Networks", *In IEEE/CreateNet BASENETS.* October 2006.

[21] D. Lymberopoulos, A. Barton-Sweeney, and A. Savvides, "Sensor Localization and Camera Calibration using Low Power Cameras", *Technical report, Yale ENALAB Technical Report 08050.* 2005

[22] D. Stephan Hengstler and H.K. Aghajan, "Wisnap: A wireless image sensor network application platform", *In 2nd International Conference on Testbeds & Research Infrastructures for the DEvelopment of NeTworks & COMmunities (TRIDENT- COM 2006), Barcelona, Spain*. *IEEE*, *March 1-3, 2006*.

[23] Georgiy Pekhteryev, Zafer Sahinoglu, Philip Orlik, and Ghulam Bhatti, "Image transmission over IEEE 802.15.4 and zigbee networks*", In IEEE International Symposium on Circuits and Systems (ISCAS 2005), volume 4, pages 3539–3542*. 23-26 May 2005.

[24] I. F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks", *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, August 2002.

[25] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A wireless sensor network testbed", in *Proc. 4th International Conference onInformation Processing in Sensor Networks (IPSN '05)* , pp. 483–488. April 2005.

[26] Omnivision Technologies Incorporated, OV6620 Single-Chip CMOS CIF Color Digital Camera *Technical Documentation at http://www.ovt.com/*

[27] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The cricket location-support system", In MobiCom, 2000.

[28] R. Y. Tsai, "A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses", *IEEE Journal of Robotics and Automation*, vol. RA-3, pp. 323-343. August 1987.

[29] D. Lymberopoulos and A. Savvides*, "XYZ: A Motion Enabled Power Aware Sensor Node Platform for Distributed Sensor Network Applications",* Information Processing In Sensor Networks, 795-825. April 2005.

[30] G. Welch and G. Bishop, "An Introduction to the kalman filter", Technical Report TR 95-041, Computer Science, UNC Chapel Hill, 1995.

[31] C. Randell, C. Djiallis, and H. Muller, "Personal position measurement using dead reckoning", *In Proceedings of The Seventh International Symposium on Wearable Computers.* October 2003.

[32] P. Kulkarni, D. Ganesan, P. Shenoy, "The case for multi-tier camera sensor network", *In Proceedings of the ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Stevenson, WA, USA,* June 2005.

[33] J. Feng, S. Megerian, and M. Potkonjak, "Model-based calibration for sensor networks", *Sensors*, pages 737 – 742. October 2003.

[34] J. M. Ortega and W. C. Rheinboldt, "Iterative Solution of Nonlinear Equations in Several Variables*", New York: Academic Press*, 1970

[35] D.G. Taylor and Song Li, "Damped Gauss-Newton Method **for** Direct Stable Inversion of Continuous-Time Nonlinear Systems", *Industrial Electronics Society, 2003. IECON '03. The 29th Annual Conference of the IEEE.* November 2003.

[36] Zigbee reference documents at http://www.jennic.com/jennic_support/. January 2007

# ACKNOWLEDGEMENTS

1. Members of the thesis committee, Dr. Weihua Sheng (Advisor), Dr. Qi Cheng and Dr. Sohum Sohoni for their invaluable guidance and support.
2. Department of Electrical and Computer Engineering at Oklahoma State University, Stillwater.
3. Research team at the ASCC laboratory, Oklahoma State University, Stillwater.
4. Mr. Sai Venugopal Lolla, (PhD), Department of Mechanical and Aerospace Engineering, Oklahoma State University, Stillwater.
5. My family and my friends for being my moral strength all through.

# APPENDICES

## APPENDIX A

Description of the experiment procedure

A VB.NET application is developed for data acquisition while implementing the calibration algorithm on a real time three node camera sensor network. The various forms of VB.NET application are shown in the Figures A and B.

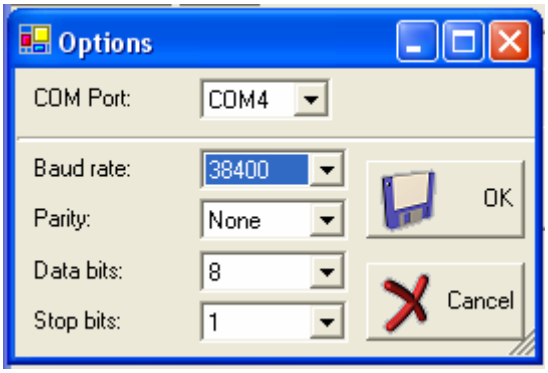



Figure A: Form to select the serial port and its parameters.

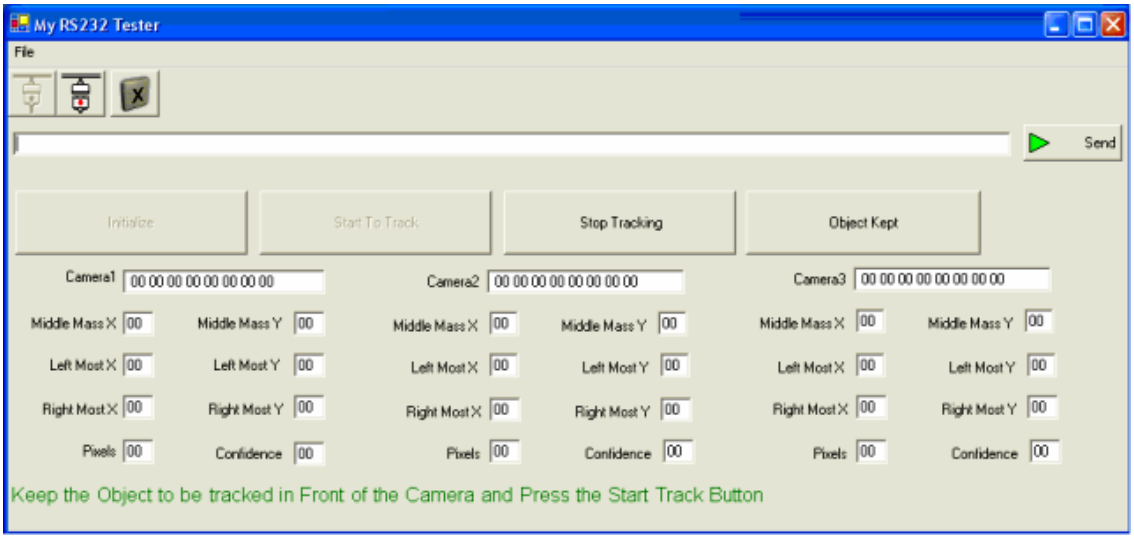



Figure B: Form to collect the data from the coordinator.

Initially, connect three cameras to three jennic modules as shown in the Figure 3.5 and switch 'On' the coordinator module. Now, switch On the end-device modules and allow them to connect to the coordinator. Connect the coordinator to the PC using a RS 232 cable.

The coordinator module turns each of its on-board LED's to 'On' one by one for each end-device that joins the network. Whenever an end-device joins the network, the coordinator assigns an 8 bit unique ID to it. Whenever, the end-device transmits a data packet to the coordinator, it prefixes this unique ID to the data packet. This enables the PC application to differentiate between data received simultaneously from various nodes.

Once all the devices are in the network, run the VB.NET application. On the main form that appears, click on "Connect" button on the main menu. The form shown in Figure A appears. On this form, select the serial communication parameters required to establish a successful communication link between the PC and the coordinator module. After selecting the parameters and pressing 'Ok' brings to focus, the form shown in Figure B.

This is the main application form that sends commands to the end devices and also stores the data received from each of the end device. The VB.NET uses the coordinator as a means to transmit command to the cameras of the nodes in the network. In order to transmit a data packet containing the command only to a specific node, the application attaches the unique 8 bit ID of the destination node to the command data packet. The coordinator is programmed to transmit the data it receives from the PC to the specified node. If the coordinator receives a data packet from any of the nodes, it directs the VB application.

To start with, click on the "Initialize" button on the main form. This causes the coordinator to send a reset command to all the end devices. The end devices, on receiving this command reset their respective cameras. The reset command is followed by an auto adjust command that causes the cameras to adjust to the light conditions in the surrounding environment. Once the above process is finished, place the target in front of the all the three cameras and click on the "Object kept button." Clicking this button causes the cameras to go into the poll mode wherein they start to track the target's blob. Now place the target at its first location such that it falls in the field of view of both the reference nodes. Now click on the "Start to Track "button. This causes the coordinator to send track command to the cameras. The cameras, in response, send a packet containing the 'X' and 'Y' coordinates with respect to their bounding box, the x and y coordinates of the image at its left most and right most positions in the bounding box. The location number of the target for that node, node 'ID' and the X coordinates of the blob are stored to a text file for every received reading. After taking a sufficient number of readings at this position, click on the "Stop Tracking" button to temporarily stop the tracking. Now,

move the object to its next predefined position such that it again falls in the field of view of both the cameras and repeat the above process and record the readings. Now, place the object alternatively at three distinct locations in the field of view of the camera node 3 which has to be calibrated and record the readings at each location. After finishing the above experiment, the data from text file for each node is given to a MATLAB program. This program runs the calibration algorithm on the obtained readings and calculates the position coordinates and orientation angles for all the three cameras.

APPENDIX B

Interfaces used for the simulation of the calibration algorithm

The calibration algorithm proposed in section 2.6 was simulated using Player/Stage software for a network of 13 camera sensors. Each sensor of the stage is equipped with a *blob finder* interface, a *fiducial finder* interface and an *opaque* interface. The moving target used for the calibration purpose is equipped with a *position2d* interface, a *fiducial finder* interface and an *opaque* interface. Each of the interfaces is explained below.

The *Blob-finder* interface

This model of the stage provides access to devices that detect and track a color blob of an object. This stage model mimics the CMUCAM2 tracking mechanism. The *blob finder* interface has a field of view of 60˚ and a range of 8 units. The image resolution of the *blob finder* can be customized by the user. The *blob finder* interface can track up to 5 different color channels. Whenever an object enters the field of view of the *blob finder* sensor and if the interface detects any one of the colors specified by its channels, it calculates the X and Y coordinates of the center of the detected blob with respect to its coordinate system. It can also calculate the height and width of the blob that has been detected. The resolution selected for the interface forms its coordinate system.

The format of the data returned by the *blob finder* interface is shown below

**Count:1**

**blob 0: id: 0 area: 60 X: 16 Y: 72 Left: 13 Right: 19 Top: 67 Bottom: 77**

The *Opaque* interface

This stage does not provide any model for a device which can add wireless connectivity to the sensor nodes. So in order to exchange information between various devices, the stage provides an *opaque* interface. This interface mimics a communication channel. The *opaque* interface sets up a communication channel between the player and the sensor node of the stage. Another client sensor node of the stage which wishes to communicate with the first node, also subscribe to the same *opaque* interface. So a communication channel has been established between these two nodes with the player acting as the medium as well as the carrier of the information between the sensor nodes. This interface can carry a message of size up to 1 MB. In order to transmit data, a node first builds a data packet and uses the command "opaque.SendCmd()". This command puts the data into the communication channel. In order to read the data in the channel, a node can send the command "opaque.Getdata()" to the *opaque* interface. All the data held in the memory of the *opaque* driver is read and stored in the memory of requesting node. Thus, the *opaque* interface mimics a communication link between two or more nodes.

The *Fiducial-finder* interface

The fiducial interface provides access to devices that detect coded fiducials or markers placed in the environment. It has range of 8 units and a field of view of 360 degrees. Each device using the *fiducial finder* is given a distinct ID called the fiducial ID. The fiducial interface of a device has range called the fiducial range within which the device can detect its object of interest. The object of interest must also have a unique fiducial ID in order to be detected by a *fiducial finder* of another object. Whenever the *fiducial finder*

interface of a device detects fiducial objects within its specified range, the fiducial interface returns the total number of fiducial objects that have been detected, the fiducial ID's of the detected objects, the X and Y coordinates of each detected object with respect to the center of the detecting fiducial sensor. The format of the data returned by the *fiducial finder* is shown below.

**#Fiducial (10:0)**

**Number of fiducial Items: 1**

**Item: 0 Id: 20 pose3d: X: 6.34837 Y: -3.78264,**

The *Position2d* interface

The *position2d* interface is used to control the mobile robot bases in 2D. This interface imparts mobility to a robot. This interface provides set of commands that can be used to steer the robot at required speed and turn angle in the stage world. This interface also provides commands that return the current speed and direction of the moving robot with respect to its start position in the X and Y directions. The interface can also return the current orientation of the robot with respect to its starting position while the robot is moving or still. For the calibration algorithm, the tracking object is equipped with this interface so that it can be made to move to different locations in the reference coordinate system so as to calibrate the nodes in the environment. The format of the data returned by the *position2d* interface is shown below.

**#xpos ypos theta speed sidespeed turn stall**

**2.62711 3.99419 1.19034 0.2 0 0.00349998**

Description of Simulation procedure

The simulation experiment is run by the player. The sensor nodes and the tracking target are programmed using C++. These programs allow the devices to subscribe for their respective interfaces declared in the stage configuration file. The entire simulation can be divided into two phases.

Phase I of the Simulation procedure

Phase I of the experiment is the localization of the tracking target by the two reference sensor nodes and calibration of the reference sensor nodes. The tracking target is initially placed at a location so that it simultaneously falls in the field of view of the two reference cameras. The two reference cameras take the blob readings of the target at this position as shown in Figure C

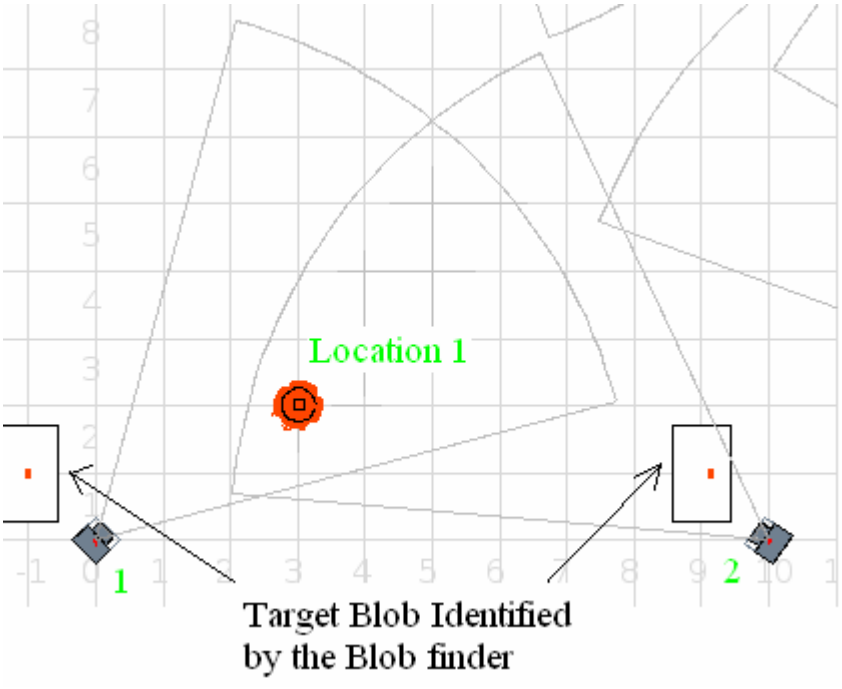



Figure C: Target at Location 1

A position vector (distance and angle) is now fed to the *position2d* interface of the tracking target. The tracking target uses this information to move to a new location. The position information should be chosen in such a way that even at this second location, the tracking target simultaneously falls in the field of view of both the reference nodes as shown in Figure D.

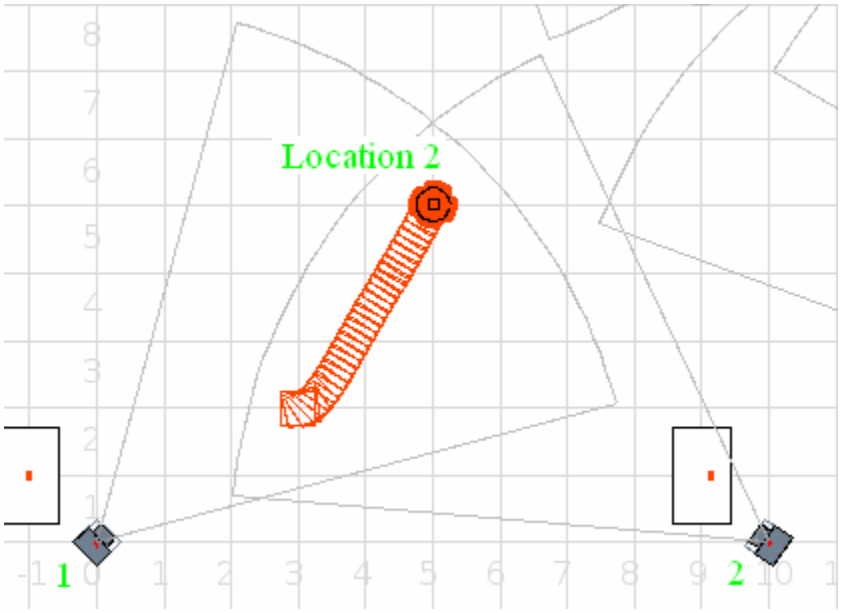



Figure D: Target at Location 2

Each sensor node now, has the blob center coordinates of the target at two different locations in their respective field of views. A single player file is written to control the tracking target as well as the reference sensor node 1 and 2.The player is now programmed to run Gauss-Newton algorithm on these readings from the two reference nodes to determine the locations of the target and the orientations of the reference nodes 1 and 2. At the end of this phase, the orientation and location information of both the reference cameras is determined. The target now uses the coordinates of the second location to determine the coordinates of its next location.

Phase II of the Simulation procedure

In Phase II of the simulation, the target moves through the world to calibrate the camera sensor nodes. While moving, data is exchanged between the target and the sensor nodes. Also, each nodes runs the calibration algorithm to determine is location and orientation information. Each of the above steps is explained in detail below.

Procedure to move the tracking target in the stage world

One assumption of the calibration algorithm is that the target should be equipped with a position sensor to record the distance and angle information it has traveled from its previous position and use this to information to calculate the coordinates of its new location. Due to the non availability of such an interface by the Player/Stage and also due to the reason that it is difficult to achieve a random motion, the target path (i.e. the points at which the target has to stop during its motion in the stage world) is decided before the experiment is actually run. However, to simulate a position sensor for the target, the location and angle information between the preset points in the world is calculated. These are fed as inputs to move the target from one position to another. At the end of phase I, the target is localized and knows its current coordinates. The target is now made to move to a new location by providing it with distance and angle information. It uses trigonometric equations to determine the coordinates of its next destination. Initially, the target moves with a speed 0.2 units/second. The target now uses the *position2d* interface information, combines it with the coordinates of its starting point to calculate its current

coordinates. It then calculates the distance information between the required destination coordinates and current location. It also determines the turn angle required to reach the destination. If the distance is greater than 0.02 units, the target turns with the required angle and moves forward with a velocity 0.2 units/second. Again it repeats the above procedure until the distance between its current location and required destination is between 0.0 and 0.02 units. At the end of the above procedure, the tracking target is approximately close to its preset location.

Data exchange between the sensor node and the tracking target

Before starting to move, the tracking target subscribes to the *opaque* interfaces of all the sensor nodes present in the stage world. Each sensor node offers two *opaque* channels. The target subscribes to each of the *opaque* interface of the sensor node number 3 to 13. Thus, two distinct communication channels are allotted between the target and each of the sensor nodes. Of the two communication channels the target uses one channel to transmit data to the sensor node and uses the other channel to receive data from the sensor node. The communication link between the target and a sensor node is shown in the Figure E

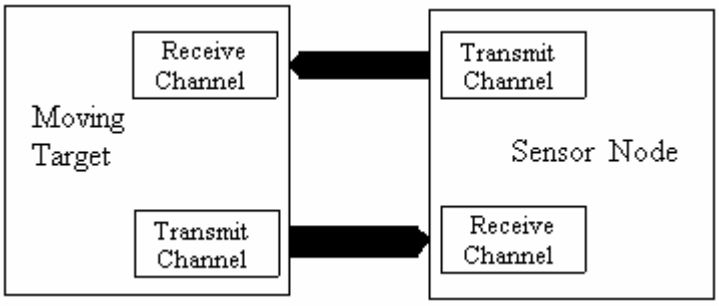



Figure E: Communication link between the target and each sensor node.

However, there is one limitation to the *opaque* interface. If a data packet transmitted by the tracking target over the transmitting channel is not read by the sensor node, it will lead to time out at the *opaque* interface on the transmitting side and the simulation will crash. Hence, care should be taken to allow data transfer over the *opaque* interface only when receiver is actively reading its data reception channel.

While moving from one location to another in the reference coordinate system, the *fiducial finder* of the tracking target is configured to search for fiducial signals from the sensor nodes in the environment. Since the fiducial range of the tracking target is equal to that of a sensor node, they detect one another simultaneously. The *fiducial finder* of a sensor node can detect the fiducials even from other sensor nodes if they are separated by a distance less than the fiducial range. However, the sensor nodes are interested only in detecting the presence of the tracking target and hence are programmed to ignore the fiducials of the other nearby nodes.

After moving to the preset location specified for the position vector, the target comes to a stop. Once it stops, the *fiducial finder* of the target scans for the presence of sensor node fiducials. There are two types of scenarios encountered whenever the fiducials of the tracking target and a sensor node detect each other. They are explained below.

Scenario I: The target is in the fiducial range of the sensor node as well as in the field of view of its *blob finder*

Once a target has been detected by the fiducial of a sensor node (say node 3), the fiducial sensor readings of the node 3 are read continuously to the determine X and Y coordinates of the position of the target relative to the center of sensor node. Any change in these coordinates between successive readings indicates that the target is moving. A moving

target is an indication that no data will be present over the communication channel and hence no attempt is made to read the *opaque* interface, which if done will lead to time out.

Whenever, the change in the X and Y coordinates of the *fiducial finder* between successive readings is '0', the sensor node conclude that the target is still. Once the sensor node has confirmed that the tracking target has stopped, it enables its *blob finder* interface check if the target is in the field of view of its camera. If the blob reading is non-zero as shown in Figure F, the senor node builds the following a data packet as shown in Figure G.

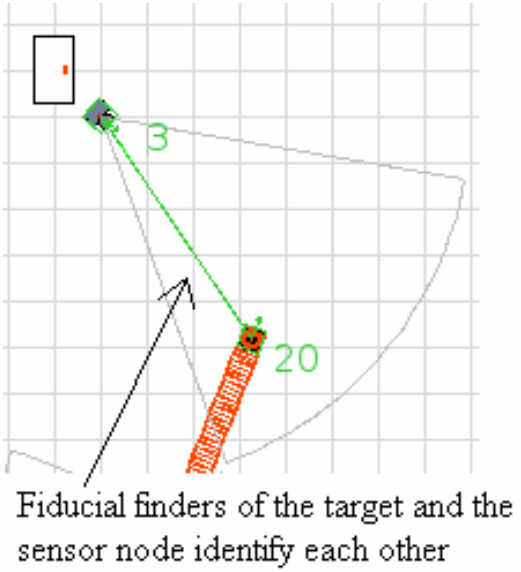



Figure F: Target detected in the field of view of sensor node 3

data count = 12

| 20 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|

The data packet transmitted to the target
by the sensor node

Figure G: Format of the data packet sent from the sensor node to the target

In the above Figure, ‘12’ represents the total packet length in bytes, ‘20’ represents the receiver’s ID (the Tracking target ID) and ‘40’ is a character indication to the receiver that it is in the field of view of the transmitter. Once the node ‘3’ sends this packet, it waits for a data packet from the target.

Simultaneously, when the target comes to a stop at its preset location, it is programmed to search for the nearby fiducials and will now detect fiducial ID ‘3’. The target now reads its receiving communication channel setup by the *opaque* interfaces of the target and the sensor node ‘3’.It reads this channel so as to verify the detection information from the node ‘3’. If the second byte of the received data packet is ’40’ the target is programmed to transmit the following data packet over transmitting channel connecting it to node 3. Figure H show the data packet received by the sensor node.

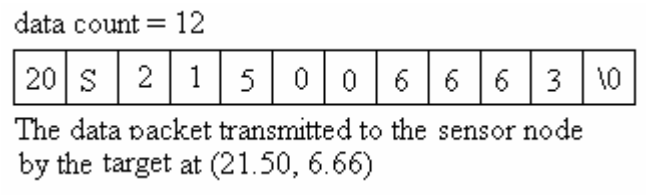
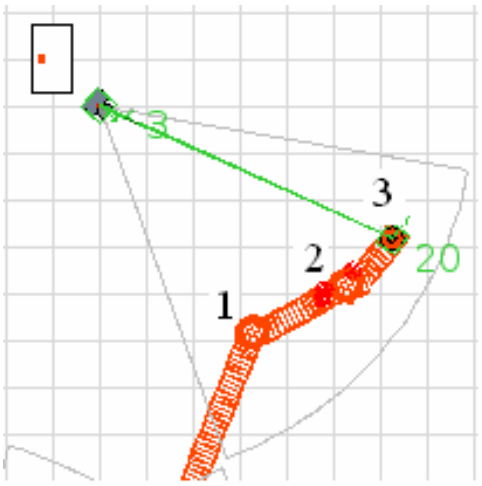



Figure H: Data packet from the target to the sensor node

In the above data packet, ‘S’ is an indication to the sensor node that the target has stopped at a location in the field of view of the *blob finder* of node 3, the next 4 bytes (bytes 2, 3, 4 and 5) are the individual digits of the X coordinate of the target’s current location and the remaining 4 bytes (bytes 6, 7, 8 and 9) are the individual digits of the Y coordinates of the target’s current location. If the stopped target coordinates are say (21.50 and 6.66), the data packet transmitted will be as shown in Figure 4.6. The target keeps a track of the number of times it has transmitted its location information to the requesting sensor node. On the reception of the above data packet shown in Figure H, node 3 rebuilds the original

X and Y coordinates of the target. It has a position counter which keeps a track of the number of positions at which the target has stopped in its field of view of its blob finder. The position counter is incremented by one every time the location information from the target is obtained. The sensor node is programmed to store the coordinates of the target's current location, the X coordinate of center of the blob of target (from its blob finder) and the position counter value into its memory. After transmitting its position information to the sensor node, the target moves on to its next predefined location and the above process repeats until the sensor node has the position information and blob values for the target for three distinct locations in its field of view as shown in Figure I.




Target at 3 distinct locations in the field of view of the camera of the sensor node

Figure I: Target at 3 distinct locations

At the end of the above procedure the sensor node will have the data in its memory as shown below.

| Position | BlobReading | X | Y | Phi |
|---|---|---|---|---|
| **1** | **67** | **7.26** | **10.14** | **-17.3228** |
| **2** | **38** | **9.26** | **11.14** | **4.17509** |

**3 21 10.25 12.13 16.6272**

The calibration algorithm requires the sensor node to capture the target information at three distinct locations in its field of view. In the simulation experiment, the target path is chosen in such a way that the target stops at three distinct locations in the field of view of the camera of each sensor node. The sensor node is programmed to run the mathematical calculations of the calibration algorithm (discussed in section 2.6.2) on the above data. Thus, it determines its location coordinates and the orientation of its camera in the reference coordinate system. It now builds a data packet carrying the above information and transmits it to the target. Let us assume that node '3' has finished running its calibration algorithm and has ended up with the coordinates (12.54, 6.54) at an angle 145.34˚ with respect to the positive X axis. This information is to be transmitted to the target. So, the sensor node now builds a data packet as shown in the Figure J.

data count = 15

| 3 | 1 | 2 | 5 | 4 | 0 | 6 | 5 | 4 | 1 | 4 | 5 | 3 | 4 | \0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Sensor Node at (12.54, 6.54) at an angle 145.34 degrees
Calibration information transmitted to the target by the sensor node

Figure J: Packet carrying the location and orientation information of the sensor node.

It is a 15 byte data packet with the byte '0' specifying the fiducial ID of the sender node, bytes 1 to 4 are the individual digits of the X coordinate of the sensor node, bytes 5 to 8 are the individual digits of the Y coordinate of the sensor node and bytes 9 to 13 are the individual digits of the orientation of the camera of the sensor node.

Simultaneously, at the target, the count of the number of transmitted locations to node '3' will have reached '3'. On reaching this count, the target reads it receiving communication channel from node '3'. Once it receives the above data packet, it rebuilds the target

information and stores it in its memory. The target now moves on to its next predefined location to calibrate the other sensor nodes in the environment. While stopping to calibrate other nodes, the target can still fall in the fiducial range of the already calibrated node for example node '3'. If the target now transmits its location to node '3' which has already been calibrated, it will accept messages from the target, this will lead to time out and the simulation will crash. So in order to prevent this, the target checks for the position counter information of each of its fiducially detected nodes. If the position counter for a particular node (node 3 in this case) is 3, the target refrains from transmitting location information over the communication channel connecting it to that particular node and hence prevents the possibility of a simulation crash.

Scenario II: The target is in the fiducial range of the sensor node but not in the field of view of its *blob finder*

Consider the Figure K

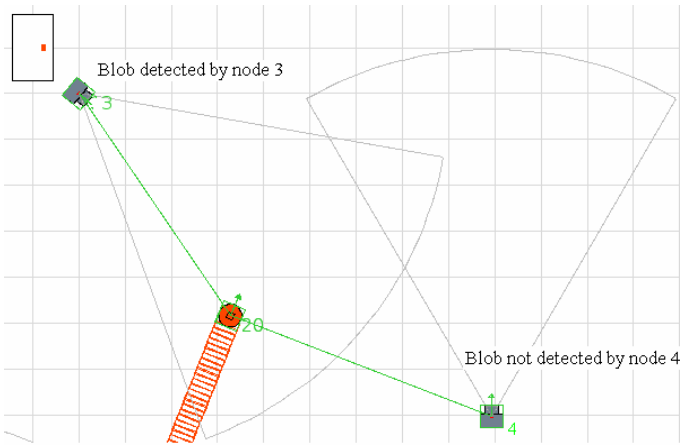



Figure K: Target blob detected by one node but not by the other.

The target's *fiducial finder* has detected the node '3' and '4' when it has stopped. Since the *blob finder* of node '3' has detected the target, it will continue with the process stated in section 4.5.2.1. However, as the target is not in the field of view of the *blob finder* of

node 4. Node 4 sends the below packet shown in Figure L to the target and waits for a data packet from the target.

data count = 12

| 20 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | \0 |
|---|---|---|---|---|---|---|---|---|---|---|---|

The data packet transmitted to the target by the sensor node 4

Figure L: Data packet sent by node 4 to the target

In the above packet, '30' is a character indicating the target that is not in its field of view. As the target detects the fiducials of nodes '3' and '4' when it has stopped, it will receive data packets from node '3' and '4'. As the data packet from node '4' gives an indication to the target that its blob has not been detected by the *blob finder* of node '4', the target sends the data packet shown in Figure M back to node '4'.

data count = 12

| 20 | M | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | \0 |
|---|---|---|---|---|---|---|---|---|---|---|---|

The data packet transmitted to the sensor node 4 by the target at (21.50, 6.66)

Figure M: Data packet from the target to the sensor node 4

In the above data packet, 'M' is an indication that the target is not transmitting its current location to the node. On receiving the above packet, the node does not increment its position counter as it has not received any position information from the target. This causes the program in the sensor node to again scan its fiducial sensor until the object falls in the field of view of its blob finder. Once this is achieved, the sensor runs the process described in section 4.5.2.1.

Following the procedures described in sections 4.5.2.1 and 4.5.2.2, the target moves through the entire environment and collects the location and orientation information from

each of the sensor nodes it encounters in its path. Finally, at end of the above simulation, the target's memory contains the position and orientation information of all the sensor nodes that it has encountered in its path. They are stored in the target's memory in the below shown format.

| CameraID | Cameral | CameraY | CameraAngle |
|---|---|---|---|
| **1** | **0** | **0** | **44.6634** |
| **2** | **10** | **0** | **134.832** |
| **3** | **14.72** | **2.2** | **128.75** |
| **4** | **17.19** | **3.91** | **127.94** |

Thus, a network of distributed camera sensor nodes can be calibrated using the algorithm proposed in Section 2.6. The simulation procedure is explained using the following flowcharts.

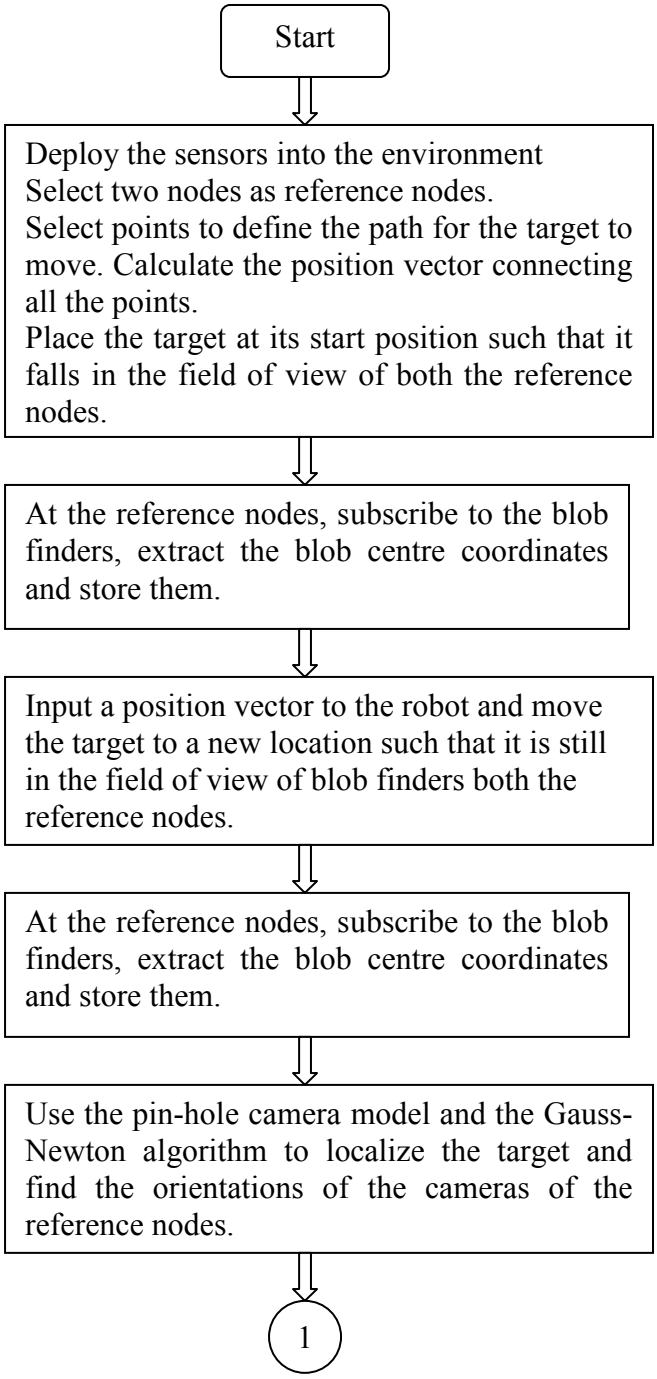



Figure: N: Flow chart for phase I of the simulation

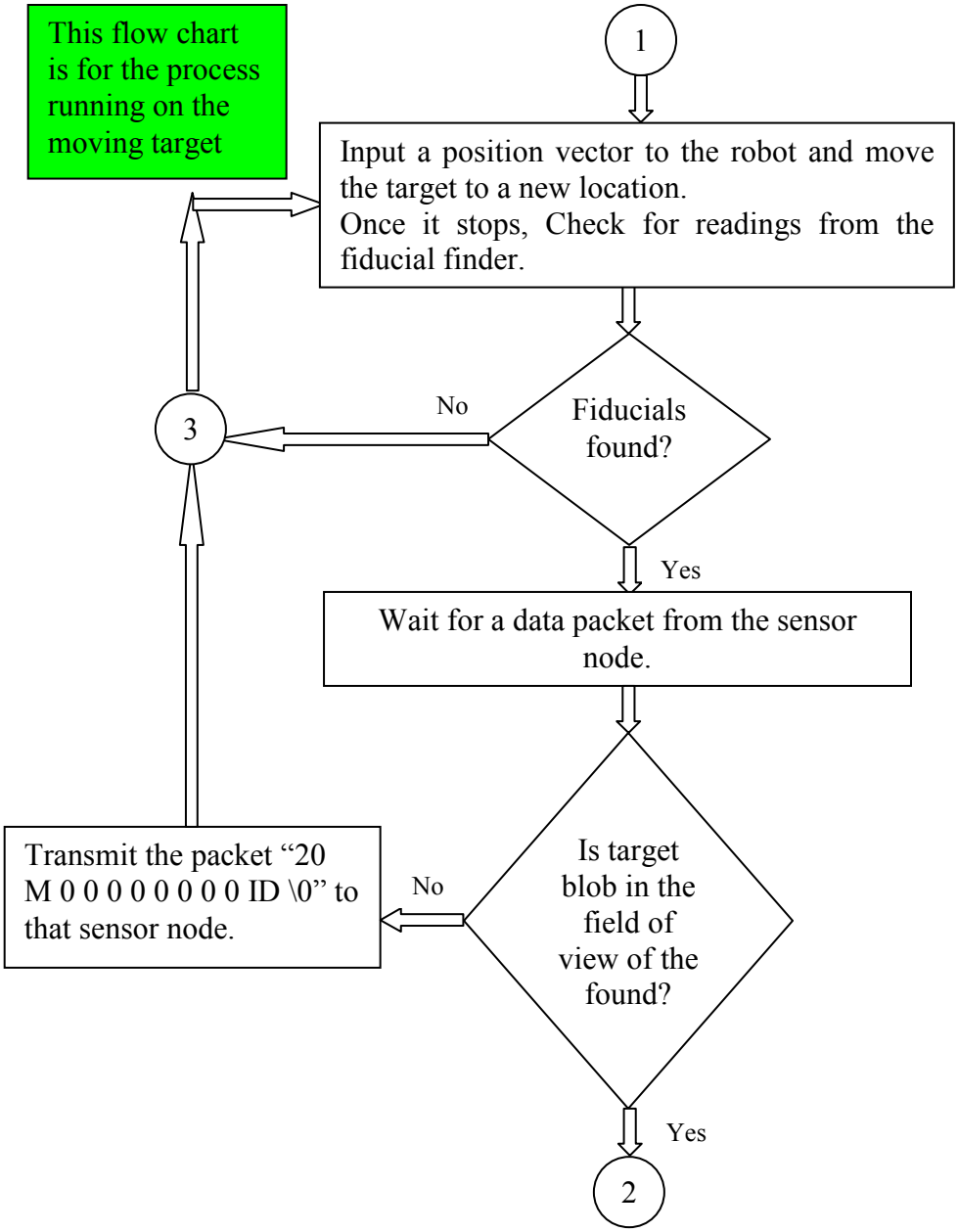



Figure O: Flow chart for the process running on the tracking target robot.

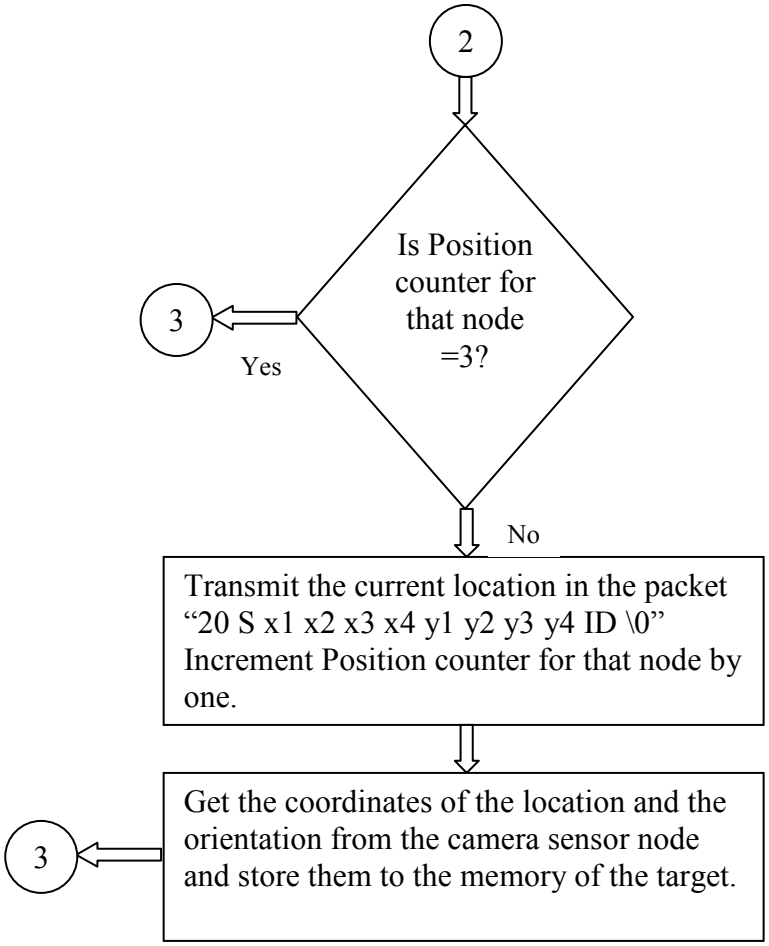



Figure P: Flow chart for the process running on the tracking target robot in phase II of the simulation.

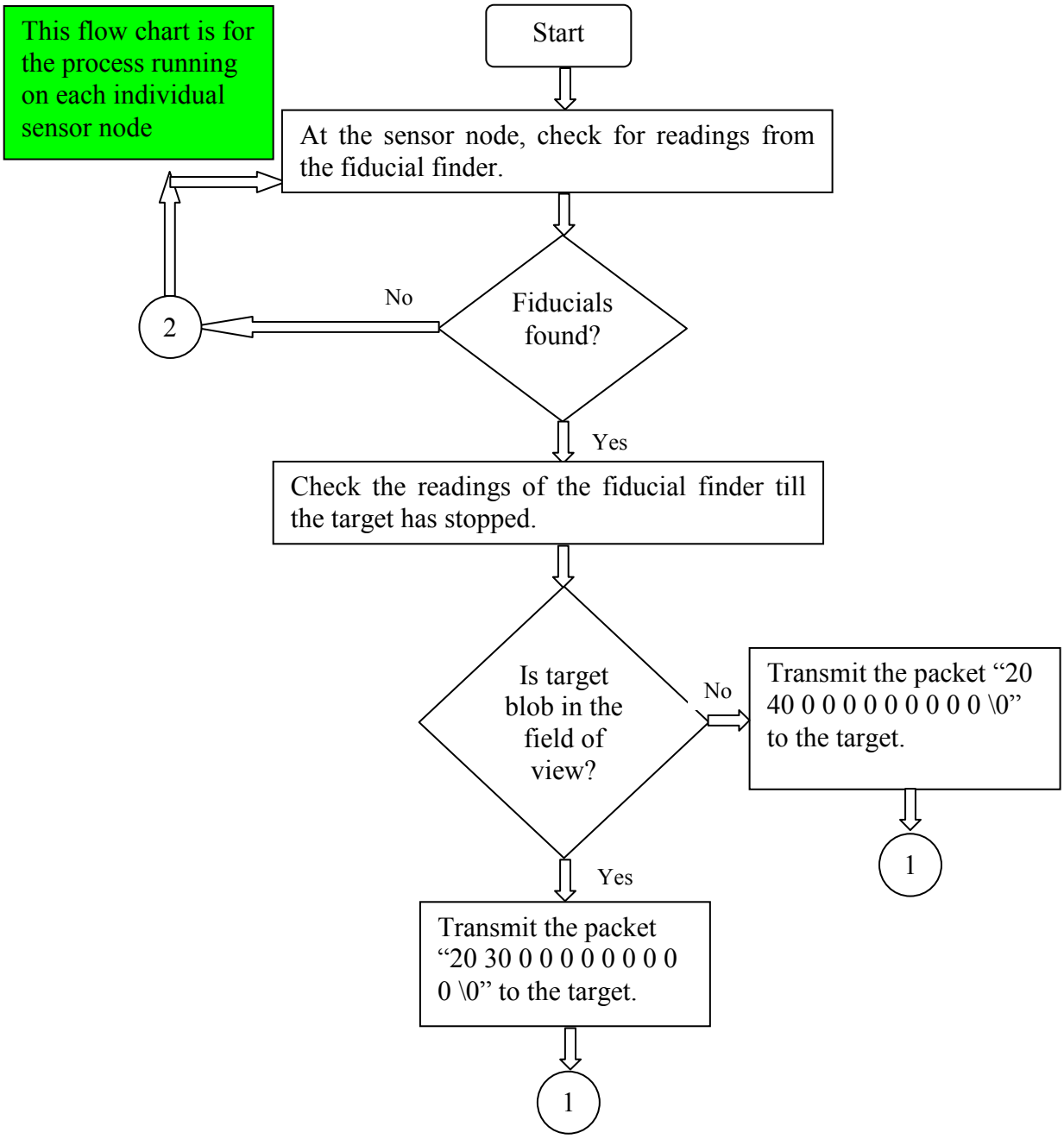
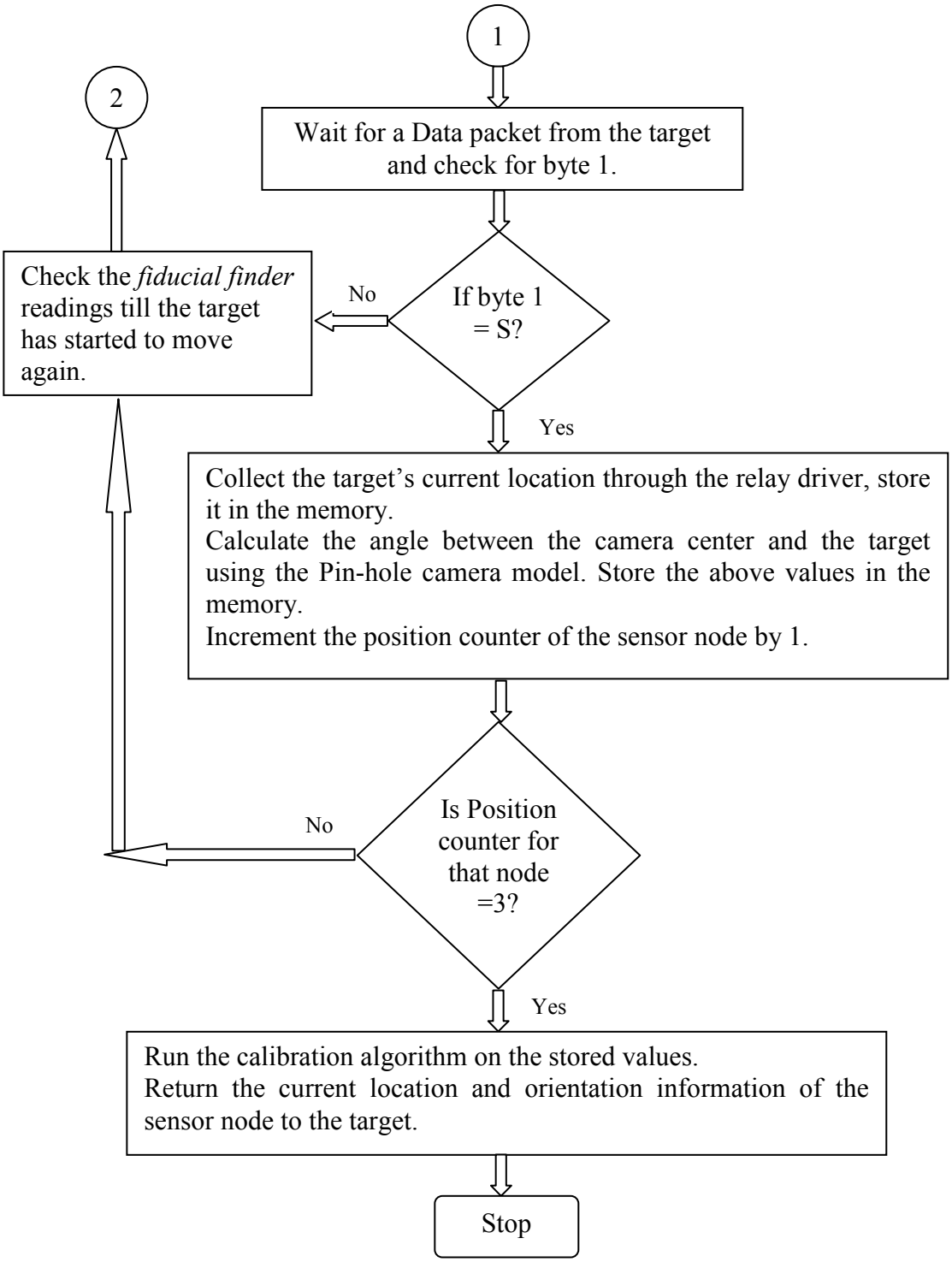



Figure Q: Flow chart for the process running on the camera sensor node in Phase II of the simulation.


1
2
Wait for a Data packet from the target and check for byte 1.
If byte 1 = S?
No
Check the *fiducial finder* readings till the target has started to move again.
Yes
Collect the target's current location through the relay driver, store it in the memory.
Calculate the angle between the camera center and the target using the Pin-hole camera model. Store the above values in the memory.
Increment the position counter of the sensor node by 1.
Is Position counter for that node =3?
No
Yes
Run the calibration algorithm on the stored values.
Return the current location and orientation information of the sensor node to the target.
Stop

VITA

Vimal Mehta

Candidate for the Degree of

Master of Science

Thesis: DISTRIBUTED CALIBRATION OF A CAMERA SENSOR NETWORK

Major Field: Electrical Engineering

Biographical:

Personal Data:

Education:
Completed the requirements for the Master of Science at Oklahoma State University, Stillwater, Oklahoma in July, 2008.

Experience:

Professional Memberships:

Name: Vimal Mehta Date of Degree: July, 2008

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: DISTRIBUTED CALIBRATION OF A CAMERA SENSOR NETWORK

Pages in Study: 82 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Scope and Method of Study: The thesis proposes algorithms to perform distributed calibration of a camera sensor network. Unlike other calibration techniques which involve non-linear computations, we propose simple techniques which can be implemented on low power processors. The concept of pin-hole camera model and generation of a reference coordinate system that are used in this thesis, were inspired from some existing work. A cooperative friendly target equipped with a dead reckoning position sensor, moves around in the environment and communicates its coordinate information at three distinct locations in the field of view of a camera sensor node. The sensor node uses the concept of slope angle of a straight line to solve for its location and orientation. This method does not require any initial guesses to calculate the external parameters of a camera. The correctness of the method was verified experimentally and its scalability for a large scale sensor network was validated through simulation.

Findings and Conclusions:

During the literature review, we came up with the following findings. Most of the state of the art calibration techniques have the disadvantage that they either require the presence of landmarks to perform calibration or those which do not use landmarks either give only the relative locations and orientations or require complex non-linear computations. The algorithms proposed in this thesis are relatively fast and involve simple mathematical calculations. Also, the proposed calibration algorithm will not require any initial guesses and hence avoids the possibility of getting stuck at a local minimum. As the expressions for the final mathematical solutions are fairly simple, they can be programmed on low power processors which are at least able to support floating point arithmetic operations.

ADVISER'S APPROVAL: Dr. Weihua Sheng