NEW POSITIONING SYSTEM FOR THE CSL ROBOTICS LABORATORY TESTBED

BY

WILFREDO LOPEZ-MORALES

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2009

Urbana, Illinois

Advisers:

Assistant Professor Dušan M. Stipanović
Professor Seth A. Hutchinson

ABSTRACT

A new positioning system is proposed for the B16 Robotics Laboratory testbed. The system is vision based and distributed. Each mobile agent is responsible for solving its individual positioning problem. The modified algorithm implements detection and identification techniques for position markers, following a template matching approach. Two modes of operation are implemented, Barycentric and Change of Axes. The new system is able to calculate position using the two methods, and 90% or more of the data points collected in each set lie within 0.05 units from true position. Heading is also calculated, but further testing is needed to fully asses its performance. The concept for the new positioning system is validated. The system achieves partial functionality and acceptable performance. However, it has not yet reached the potential necessary to replace the current system.

TABLE OF CONTENTS

1. INTRODUCTION

The B16 Robotics Laboratory located in the Coordinated Science Laboratory is equipped with an autonomous multi-agent ground vehicle testbed. It is an indoor facility whose purpose is to serve as a platform for development and testing of multi-robot control algorithms, i.e., collision avoidance and formation control. It employees machine vision, UDP communication protocols, wireless broadcast to vehicles and computing capability onboard the vehicles.

This thesis proposes a new positioning system that is based on machine vision, but is distributed among the mobile agents. The two key objectives are the exploration of this new scheme for a positioning system and a decrease in overall system latency.

The thesis is divided into five chapters structured as follows. Chapter 2 presents the underlying theory behind machine vision and the mathematics involved in both of the positioning algorithms developed. Chapter 3 describes the existing and new hardware used and explains the algorithm and its two modes of operation. Chapter 4 details the experiment, presents the results and includes a discussion of the results and observations from the experiments. Finally, Chapter 5 presents conclusions and suggestions for future work.

2. THEORY

The theory for machine vision and digital images is covered in this chapter. Also, the mathematics involved in the algorithm described in [1] is presented here, followed by a description of other testbeds located at other universities.

2.1 Digital Images

A digital image can be described as a discrete, two-dimensional array of light intensity values whose elements, called pixels, are arranged in rows and columns (*r,c*). The pinhole lens model dictates that the focal center of the lens is considered to be an ideal pinhole through which light rays enter and intersect the image plane. Under this assumption, a point *P* in world coordinates (*x,y,z*) is collinear with its projection *p* in the image plane (*u,v*) as seen in Fig. 2.1. The relationship between the image coordinate frame and pixel array lets us relate the digital image to the real world.

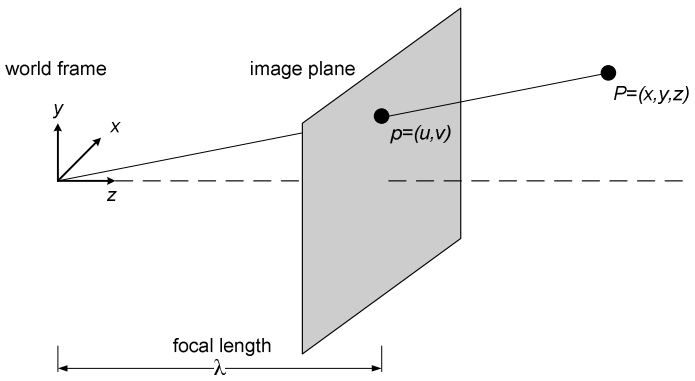


Figure 2.1: World and image coordinate frames.

2.2 Camera Calibration

Camera calibration is necessary to relate the world frame with the image array. In order to properly calibrate the camera, some parameters have to be determined. Camera parameters are of two types: intrinsic and extrinsic.

Intrinsic parameters are those that remain constant for the camera regardless of its position. These parameters are the pixel array coordinates for the principal point O and the ratios between focal length and pixel dimension f_k, defined below:

$$O = (o_r, o_c)$$
$$f_x = \frac{\lambda}{s_x}$$
$$f_y = \frac{\lambda}{s_y}$$

where λ is the focal length and s_x, s_y are the horizontal and vertical dimensions of the pixels.

Extrinsic parameters define the position and orientation of the camera relative to the world coordinate frame. The extrinsic parameters are the rotation matrices **R** and **T** defined as

$$\begin{aligned} \mathbf{R} &= \mathbf{R}_w^c \\ \mathbf{T} &= -\mathbf{R}_w^c \mathbf{O}_c^w \end{aligned} \tag{2.1}$$

where $\mathbf{R}_w^c$ denotes a transformation from camera pixel array coordinates to world coordinates and $\mathbf{O}_c^w$ denotes the world frame origin.

The coordinate transformation is achieved by solving the linear system:

$$\mathbf{x}^c = \mathbf{R}\mathbf{x}^w + \mathbf{T}$$

where $\mathbf{x}^c$ is the vector that contains camera frame coordinates and $\mathbf{x}^w$ is the vector that contains world frame coordinates; **T** and **R** are defined in Eq. (2.1).

2.3 Segmentation by Thresholding

The main goal of image segmentation in this project is to separate the image data into two regions: a region that corresponds to the objects of interest and a region that corresponds to the background. This type of segmentation results in a binary image, because pixels belong to one of only two classes.

Grayscale image thresholding is performed by choosing a gray level threshold value. Say that a dark object sits against a light background. Pixels whose gray level values are equal to or less than the threshold level belong to the object, and all remaining pixels are then classified as background. The inverse is valid for light objects against dark backgrounds. The problem with grayscale thresholding arises when choosing the proper gray level threshold value. The threshold can be specified *a priori* based on some initial observations. A histogram that displays the occurrence of each gray level value for all pixels in the digital image is used to choose an appropriate gray level threshold that separates object gray levels from background. Alternatively, the gray level threshold value can be automatically chosen with an algorithm. Some statistical information is computed from the gray level values and an automatic selection can be made based on

the gray level distribution of each image. The complete automatic algorithm is described in Chapter 11 of [1].

Color images present a more complicated problem when it comes to segmentation because color pixels have three values instead of a single gray level value. As in grayscale segmentation, there are numerous ways to approach this problem; in this thesis a deterministic approach analogous to grayscale threshold segmentation is followed.

2.4 Color Models

A color model is a mathematical model that describes the representation of colors by groups of numbers. Two of the most common color models are the Red, Green Blue (RGB) and Hue, Sat, Value (HSV) models.

2.4.1 RGB

The RGB color system was developed before the digital age and is based on human perception of color. It is an additive model in which quantities of each of the primary colors, red, green and blue (hence RGB), are summed together to represent another color. The absence of all three primary colors yields darkness or black and full intensity of the three primaries yields white. Color CRT monitors display a subset of the RGB color space called mRGB. Its primary colors are the same: red, green, and blue. Since each pixel element in a CRT monitor has a finite intensity value, the color solid of the mRGB

system is a bounded subset of the space generated by the primaries. By scaling each primary axis, the coordinates can be normalized such that the maximum value is 1. This way, the subset forms a color solid and it is called the RGB cube. Its origin (*0,0,0*) corresponds to the color black (no color intensities present) and the point (*1,1,1*) corresponds to the brightest white possible by the monitor [2].

2.4.2 HSV

The HSV model is derived from the mRGB model. Its name is results from the parameters hue, saturation, and value. By definition, any color with coordinates (R,G,B) in the color cube will have a HSV value parameter equal to the maximum of the R, G, B values. The color figure of the HSV system is shaped by the orthogonal projection of all parallel RGB cubes that form along the main diagonal from (*0,0,0*) to (*1,1,1*). These projections form a pyramid with hexagonal base and its vertex at (*0,0,0*). For each cube with main diagonal coordinates (*p,p,p*), all colors contained in the hexagonal projection will have the same value parameter [2].

2.5 Connected Components

When multiple objects are present in the digital image, it is useful to distinguish and separate them. After a digital image is segmented, the binary image does not contain information of individual objects; rather, it has the pixels of all seen objects. An object can be identified in a binary image as a connected component. A connected component

is any group of contiguous pixels such that every pixel is in direct contact to its neighboring pixels, and there is a continuous pixel-to-pixel path that connects all pixels within the group. Direct contact with neighbor pixels can be defined in two ways: 4-connected and 8-connected. Pixels that are 4-connected to the pixel of interest are those that are directly above and below and before and after. 8-connected pixels also include those that are diagonally adjacent. After a connected component is identified, all the pixels contained in it are labeled the same such that the object is identified; that is, each component has a unique label. Many component labeling algorithms exist; this thesis utilizes the one described by [1].

2.6 Moments

Once objects have been identified and labeled in the digital image, moments can be computed in order to acquire some information on the shape and size. The *i,j* moment for the k^{th} object, denoted by $m_{ij}(k)$, is defined by:

$$m_{ij}(k) = \sum_{r,c} r^i c^j I_k(r,c)$$

The order of a moment is defined as the sum $i+j$, and $I_i(r,c)$ is an indicator function defined as

$$I_i(r,c) = \begin{cases} 1 & : \text{ pixel } r,c \text{ is contained in component } i \\ 0 & : \text{ otherwise} \end{cases}$$

where *r, c* denote row and column numbers for the pixel.

It is of high interest to define an object's center of mass. The center of mass of an object is defined as a the point $(\bar{r},\bar{c})$ such that if all that object's mass were to be concentrated at $(\bar{r},\bar{c})$, the first moments would not change. Thus,

$$\sum_{r,c}\bar{r}_i I_i(r,c)=\sum_{r,c} rI_i(r,c) \quad \Rightarrow \quad \bar{r}_i=\frac{\sum_{r,c} rI_i(r,c)}{\sum_{r,c} I_i(r,c)}=\frac{m_{10}(i)}{m_{00}(i)}$$

$$\sum_{r,c}\bar{c}_i I_i(r,c)=\sum_{r,c} cI_i(r,c) \quad \Rightarrow \quad \bar{c}_i=\frac{\sum_{r,c} cI_i(r,c)}{\sum_{r,c} I_i(r,c)}=\frac{m_{01}(i)}{m_{00}(i)}$$

A moment calculated with respect to the object center of mass has characteristics that are invariant with respect to translation. These moments are called central moments and are very useful when determining shape and size characteristics of the object. The *i,j* central moment for the k^{th} object is defined by:

$$C_{ij}(k)=\sum_{r,c}(r-\bar{r}_k)^i(c-\bar{c}_k)^j I_k(r,c) \tag{2.2}$$

where $(\bar{r}_k,\bar{c}_k)$ are the coordinates for the center of mass of the k^{th} object.

The orientation of an object will be defined as the direction of the axis that passes through the object in such a way that the second moment of the object about that axis is minimal. Then for any given line in the image:

$$L=\sum_{r,c} d^2(r,c)I(r,c)$$

where *d(r,c)* is the distance from pixel (*r,c*) to the axis.

The objective is to minimize L with respect to all possible lines in the image plane. The authors in [1] describe the method to do this and only the final result is presented here:

$$\tan 2\theta = \frac{2C_{11}}{C_{20} - C_{02}}$$

where C_{11}, C_{20} and C_{02} are defined in Eq. (2.2).

2.7 Coordinate Transformations

Coordinate transforms can be used to scale, translate and rotate objects within a coordinate system. It is particularly useful when referencing one coordinate system in terms of another. In machine vision, the relationship between image coordinates and world frame coordinates is dictated by a change of axes, or coordinate transform. This change of axes can be achieved with three operations: scaling, translation and rotation. Scaling refers to a change in proportions between the reference frame (x, y) and new frame (x', y'). It is performed by multiplying the matrix with scale factors α_x, α_y:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & 0 & 0 \\ 0 & \alpha_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation introduces a displacement between the coordinate axes. Displacement in direction and magnitude δ_x, δ_y are performed by multiplying the matrix below:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \delta_x \\ 0 & 1 & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation is the motion around a fixed pint, in our case the coordinate axes origin, by an angle ϕ and is achieved by multiplying the matrix:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

These operations can be combined into one transformation

2.8 Barycentric or Areal Coordinates

Barycentric coordinates are a form of homogeneous coordinates defined by the vertices of a simplex. Also called areal coordinates, barycentric coordinates locate points relative to existing points, rather than to an origin. Let T be a triangle defined by the three vertices v_1, v_2, and v_3. Let p be any point located within triangle T. Point p can then be written as the weighted sum of the three vertices:

$$p = \lambda_1 v_1 + \lambda_2 v_2 + \lambda_3 v_3$$

where λ_1, λ_2, and λ_3 are the barycentric coordinates and exist inside the set [0,1]. The conversion from Cartesian coordinates to barycentric coordinates is a linear transformation problem. The Cartesian coordinates of $p = (x_p, y_p)$ can be written as:

$$\begin{aligned} x_p &= \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 \\ y_p &= \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3 \end{aligned} \tag{2.3}$$

where (x_1, y_1), (x_2, y_2) and (x_3, y_3) are the Cartesian coordinates for the vertices v_1, v_2 and v_3 of triangle T. Barycentric coordinates λ_1, λ_2, and λ_3 must satisfy [3]:

$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \tag{2.4}$$

Rearranging Eq. (2.4) we have:

$$\lambda_3 = 1 - \lambda_1 - \lambda_2 \tag{2.5}$$

Substituting Eq. (2.5) into Eq. (2.3) yields:

$$\begin{aligned} x_p &= \lambda_1 x_1 + \lambda_2 x_2 + (1 - \lambda_1 - \lambda_2) x_3 \\ y_p &= \lambda_1 y_1 + \lambda_2 y_2 + (1 - \lambda_1 - \lambda_2) y_3 \end{aligned}$$

After rearranging:

$$\begin{aligned} \lambda_1 (x_1 - x_3) + \lambda_2 (x_2 - x_3) &= x_p - x_3 \\ \lambda_1 (y_1 - y_3) + \lambda_2 (y_2 - y_3) &= y_p - y_3 \end{aligned} \tag{2.6}$$

Equation (2.6) can be written as the linear transformation:

$$\mathbf{R} \cdot \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \end{bmatrix} - \begin{bmatrix} x_3 \\ y_3 \end{bmatrix}$$

where matrix **R** is defined as:

$$\mathbf{R} = \begin{bmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{bmatrix}$$

Since v_1, v_2, and v_3 are vertices of triangle T, they are not collinear and ($v_1 - v_3$) and ($v_2 - v_3$) are linearly independent. Hence, **R** is nonsingular, invertible and

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \mathbf{R}^{-1} \begin{bmatrix} x_p - x_3 \\ y_p - y_3 \end{bmatrix} \tag{2.7}$$

2.9 Other Testbeds

Some investigation was done to learn about existing testbeds and their systems. Several testbeds where found to be operational in different universities around the nation. A brief description of some of these testbeds follows.

University of Pennsylvania's experimental testbed for large multi-robot teams is an indoor test facility that uses custom vehicles on the ground and suspended in air. Vehicles carry an onboard computer, sensors and wireless communication capabilities. Localization of agents is done with a custom, low-cost positioning system. Vehicles carry an LED display, each illuminated with a unique 8-bit pattern. Overhead cameras are able to detect and track LED displays, and process information to calculate position and pose of robots. Uncertainty measurements are computed with a Kalman filter, and vision information combined with onboard telemetry provides ground-truth data [4].

Caltech's multi-vehicle wireless testbed is an indoor experimental area. Its hardware setup is very similar to the B16 robotic testbed. Its vehicles are equipped with onboard laptop computers, and a command PC handles all wireless transmission, global information, and commands to vehicles. Positioning is done with an overhead vision system comprising four ceiling-mounted cameras that look down on the test arena. Vehicles carry monochrome templates on top, with markings that allow cameras to identify vehicles and compute position and orientation. The vision system has a dedicated computer to do all image processing and data is sent wirelessly [5].

MIT's Aerospace Control Lab developed two testbeds: one indoor, the other outdoor. The indoor testbed works with ground and aerial vehicles. Rovers play the role of multiple agents to be controlled while the blimp acts as a supervisory, surveillance vehicle overhead. All vehicles have equal processing capabilities with onboard laptops processing sensor data, and a ground station handles heavy computing. Positioning data is acquired with a commercial off-the-shelf (COTS) meteorological system. The system is highly accurate and highly expensive [6]. The outdoor testbed consists of R/C airplanes fitted with COTS autopilots. Wireless vision is employed for position verification and user feedback.

MIT's RAVEN is an indoor, rotary wing, aerial, multi-vehicle testbed. Vehicles are COTS quad-rotors without any sensing capability. Positioning is also done with a central, metrology motion capture system. This system is very expensive, but highly accurate [7].

Stanford University's STARMAC is an outdoor testbed of rotorcraft aircraft vehicles for multi-agent control. Vehicles are COTS with Bluetooth capabilities and dual PIC for onboard sensor handling. Positioning is done with differential GPS measurements, provided by onboard GPS receivers and a ground station handling all position computations and wireless communications [8].

University of California – Berkeley's BEAR is an outdoor testbed for mobile aerial agents. Industrial R/C helicopters with onboard computers and sensors serve as mobile agents. Positioning is done by onboard GPS receivers along with inertial measurement units. Vision is used for object detection [9].

University of Washington's ARCS testbed is a small, indoor testbed that uses COTS robot agents. All heavy computed is done offboard. The positioning system is based on vision. A single, ceiling-mounted webcam runs Carnegie Mellon CMVision software. Robots carry colored templates on top; the vision system is able to identify, localize and compute heading. Information is sent wirelessly to agents [10].

Oklahoma State University's COMET is an outdoor testbed. Its mobile agents were built from modified COTS R/C trucks. They carry onboard embedded computers with the same computing capabilities as desktop PCs. The positioning system consists of GPS receivers onboard each mobile agent. It is assisted by onboard firewire cameras used to identify neighboring agents. Each agent carries boards with NameTag patterns that can be seen by other agents. Inertial measurement units also provide positioning information; however, GPS data is considered as correct [11].

University of Illinois' HoTDec is an indoor testbed. Hovercrafts serve as mobile agents, equipped with single board computers running the Linux kernel. Its positioning system consists of six firewire cameras mounted to the ceiling and arranged in pairs, each connected to a Linux box. Robots carry templates on top, with unique color patterns that

provide identification and orientation information to the image processing software. Processed image information is sent over the wireless network, Bluetooth or IEEE 802.11, to agents [12].

3. METHODOLOGY

Chapter 3 first describes the current testbed configuration. Both hardware and software aspects are briefly discussed. Then, the proposed positioning system is presented and discussed.

3.1 Hardware

The robotic testbed is an indoor facility located in room B-16 in the basement of the Coordinated Science Laboratory. The test area is approximately 14 feet by 18 feet, shared with an office space. The nature of its location presents some restrictions on modifications that can be done to the space. All electronic hardware used for the current positioning system and for the robots is described in the following subsections.

3.1.1 Positioning system

The current positioning system includes four USB cameras mounted on the ceiling, each connected to its own computer. The computers are Dell Precision desktops running Microsoft Windows XP. Each desktop runs an instance of the same image processing program built in LabVIEW. All four computers communicate via a local area network. One of the four computers also serves the role of command PC. This command PC also runs the client software that broadcasts vision data wirelessly to the robots. Robots carry an identifier template on top that allows for recognition by the vision system. This

template carries a set of dots placed into two regions that provides the vision system with enough information to uniquely identify and determine the pose of each robot. Lighting to the testbed is supplied by construction-type halogen lamps. These lamps were chosen over the existing white fluorescent tubes in the room because they provide a more uniform light source needed for the vision system.

3.1.2 Robot agents

The robots were designed by Daniel J. Block and Mark W. Spong and built by Juan S. Mejia and Chad R. Burns. They follow the same platform used for the GE423 Mechatronics Laboratory course [13]. The robots are nonholonomic, four-wheel, ground vehicles shown in Figs. 3.1 - 3.4. The onboard processing is done by the Texas Instruments TMS320 C6713 DSP running at 225 MHz with a daughter card. Sensors communicate via an I^2C bus and the robot's wireless communications link is supplied via an added wireless radio card. The primary sensors attached include optical encoders on the DC motors that run the wheels and a solid-state rate gyro. Other sensors supported include, but are not limited to, infrared and ultrasonic ranges, compass, and color camera.

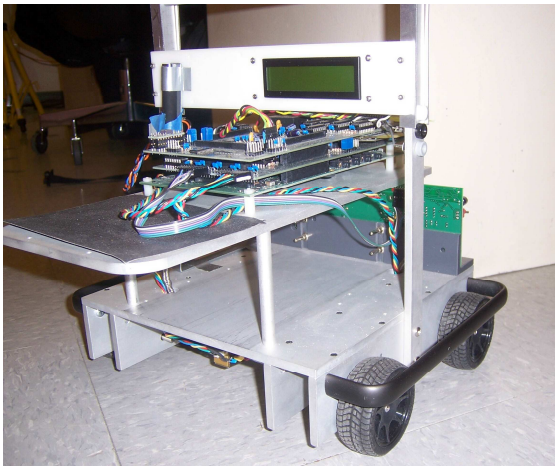


Figure 3.1: Test vehicle (back view).

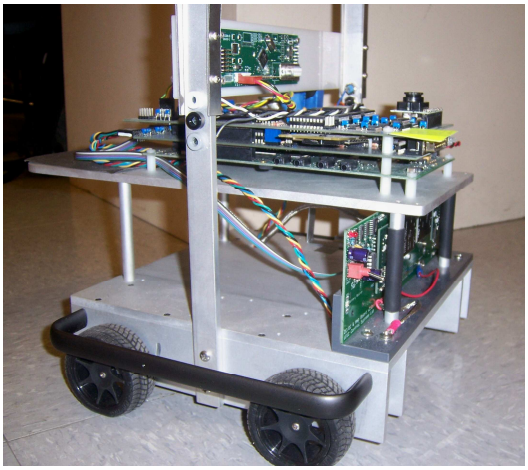


Figure 3.2: Test vehicle (front view).

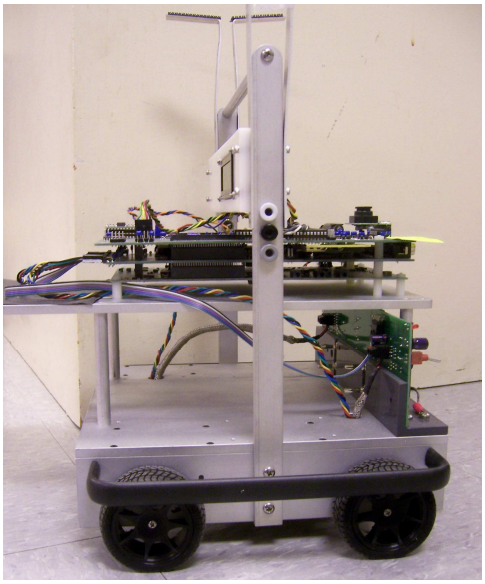


Figure 3.3: Test vehicle (right side view).

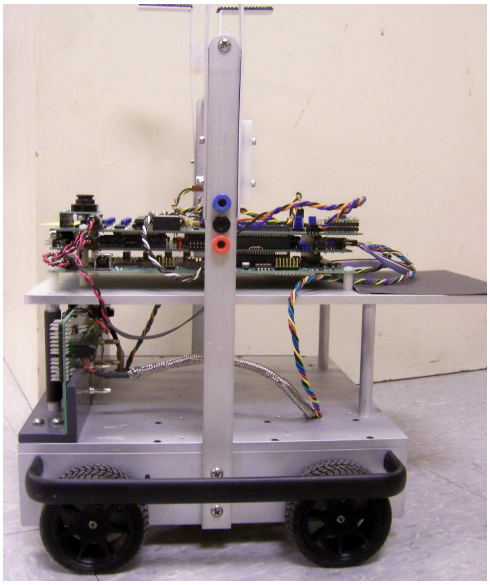


Figure 3.4: Test vehicle (left side view).

3.2 Software

Three software packages operate within the testbed. The first addresses the image processing for the positioning system. The second serves as the GUI for communication and command of the robots. The third runs locally on each robot and is tailored to the specific task desired for the robot.

3.2.1 Positioning system

The USB cameras mounted on the ceiling interface with the LabVIEW program. The vision program consists of several sub-VIs that perform the necessary processes to extract the positioning information from the images. The image is first black-and-white thresholded at a level that is manually chosen by the user and can be adjusted online. Thresholded images then pass through a blob detection algorithm. This sub-VI identifies the objects seen and makes the positive or negative identification of a template. Once a template has been positively identified, its properties are recorded and passed to an object array for transmission. Robot number is determined by the number of dots inside the main region of the template. Its center of mass is taken as the main region's center of mass. Smaller dots around the main region are used for calculating robot orientation. A vector is drawn from the center of mass to the middle of the closest pair and its orientation with respect to the image coordinate frame is recorded as the robot's orientation. All information is organized into clusters that form an array. The data is transmitted via UDP to the Qt program for display and broadcast. The loop is repeated

for each frame captured. The vision system information is considered as ground truth. Exact measurements have not been recorded, but position deviations at standstill appear to be 0.01 feet and angle deviations approximately 0.3 degrees [14].

3.2.2 Qt communications and command program

The command PC also serves as the GUI console. This application was written in C language with the Qt library by [14]. Its function is threefold: it provides the user with a GUI that displays position and orientation of all seen robots, it allows the user to send position commands by clicking anywhere within the workspace, and it sorts out all the positioning information gathered by the four cameras. All data is sent wirelessly to the robots in two types of packets: position command or ground truth information. When the user clicks on the GUI map, a data packet is sent to the robots with the desired coordinates. The robot agents then follow their uploaded control scheme and take necessary action to move to the desired position.

3.2.3 DSP program

The DSP is programmed in Texas Instruments' DSP/BIOS environment. It is designed for handling time-sensitive operations, and programming is done in Code Composer Studio (CCS) compiler using C language. In contrast to its use in standard C programming, the `main()` function is used only for initialization of sensors, chips, devices and global variables, not to contain the principal code. The DSP/BIOS kernel is

configured to execute a periodic function every millisecond. This function starts the analog-to-digital converter and activates a hardware interrupt (HWI). When HWI_7 is active, the DSP executes the `ADC_INT7_Func` function. It is in this function that the main code resides. All functions were taken from the Mechatronics Laboratory course [13]. The focus of this thesis lies on the function `userProcessColorImageFunc_laser()`, from now referenced as uPCIF, inside the *user_ColorVisionFuncs.c* file.

Some prior processing to the raw image data is done before the uPCIF has access to it. The onboard camera captures an image that is 288 rows by 352 columns of 8 bit pixel data every 0.04 seconds. Due to its large size, the image has to be stored in the external SDRAM memory instead of the DSP internal RAM. The external SDRAM runs at a slower speed than the DSP and this accounts for some delay when accessing the image array. It takes the DSP approximately 10 ms to scan through the whole image; that leaves time for only three passes before the next image is received from the camera. For this reason, the image is scanned once and compressed by a factor of four resulting in an image of 72 rows by 88 columns. The compressed image is faster to scan and leaves more time for other tasks to be performed by the DSP. uPCIF receives this compressed image and is able to perform more than the original four passes.

The code in uPCIF implements the algorithm discussed in Chapter 11 of [1]. The color image is first converted from RGB, the format in which the image is originally received from the camera, to HSV. An interesting property of the HSV color system led to its

selection over RGB. In HSV space, a color defined by its *saturation* and *hue* parameters can have a wide range of luminance values. This behavior is desirable, in hopes that it permits a more relaxed definition for a color. When identifying colors in a digital image, this translates to some compensation for some of the lighting variations encountered. The algorithm then thresholds and separates the objects in the image. It looks for the largest object of a predetermined color and extracts its position and orientation within the image coordinate frame.

3.3 Proposed New Positioning System

After the development of the B16 Robotic Testbed, some problems and areas for improvements were documented. Among these, the overall system latency was the most critical. It is well known that for stable dynamic systems, delays in updates of state observations can affect the stability properties. The B16 platform is used in the testing of control laws for collision avoidance and formation control. The delay problem may cause false failures to the experiments and hence yield false results for current investigations. Although some algorithms have been implemented that take into account the delays and can be run on top of control algorithms [14], it is still desirable to reduce delays as much as possible. The overall system delay was measured to be 100 to 400 ms [14]. The primary suspect for latency is the wireless broadcast of ground truth data. Instead of debugging the system, it was proposed to develop an alternate positioning system that does not follow the same communication channel path and that has a more decentralized architecture.

The new system should be low-cost, scalable and reliable. The proposed system requires minimal hardware acquisition and configuration. It also takes advantage of the onboard DSP computing capability. After doing some research into other testbeds developed by universities, it seems that the positioning system developed in this work has not been attempted before. Instead of having a dedicated system to manage all ground truth information, the system proposed in this thesis distributes the task of positioning at an individual level. Each robot is responsible of calculating its own position. Robots were fitted with onboard cameras and a grid of markers was constructed on the ceiling tiles. The markers provide coordinate and heading information to the robots' positioning algorithm.

3.3.1 New hardware

Onboard color cameras were installed on the robots, in a horizontal position pointing straight at the ceiling. The cameras are model BB048 ¼ inch color cameras with the OmniVision 6620 chip embedded, shown in Fig. 3.5. The cameras are fully supported by the hardware and software of the robots and are the same model used by the Mechatronics Laboratory robots [13]. They feature an I^2C interface and output a 352 x 288 pixels array of 8-bit data.

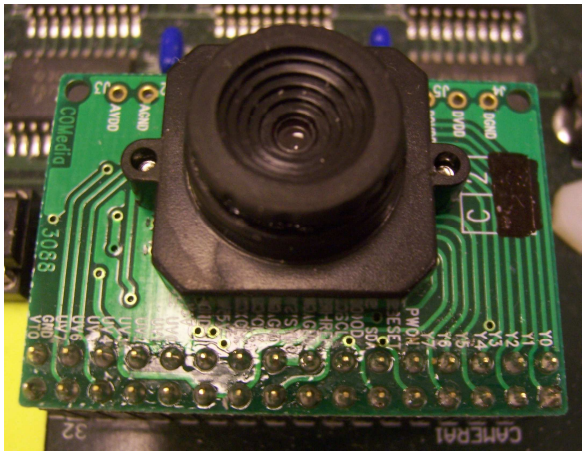


Figure 3.5: OmniVision 6620 chip embedded with 1/4 inch color camera.

The ceiling was fitted with markers arranged into an array, shown in Fig. 3.6, that lays out the workspace for the testbed. The two-foot spacing between the centers of adjacent units was chosen because the ceiling tiles are two feet wide. The marker was designed with the current robot templates in mind. The current templates provide information on robot identification, center of mass and orientation. These new markers, shown in Figs. 3.6 and 3.7, will not have to provide identification information, for each robot is manually identified by its own configuration. Markers follow a symmetrical, rectangular shape to help maintain a consistent overall center of mass. Having a constant center of mass is important because this point will be used for all further positioning calculations and should remain stable under all conditions. The markers break down into two rectangular regions, each dedicated to *x*- and *y*-coordinates. Each region is of a different background color and contains within it colored dots displaying a binary number corresponding to the marker location within the grid. The two colors chosen were bright red and bright green. These choices make good contrast against the ceiling tiles and other colors and shadows found in the ceiling. The rectangle shapes that make the markers provide a major axis that dictates the direction of the orientation vector. The most

significant bit (MSB) from the binary number is identified with the relative position of the red and green rectangles. The binary number is read along the major axis of each rectangle and, with MSB known, it can then be converted to decimal base. The dimensions for the markers were determined experimentally. Several trials were performed; the current markers were found to be adequately sized such that the cameras can recognize all regions and dots more consistently. The dimensions for the markers are 12 inches long by 10 inches high. Dots are elliptical and measure 3 inches by 2 inches (green), and 2 inches by 1 inch (red). The difference in dot size is due to the sensitivity of the recognition scheme to different colors. Green shapes need to be larger for better recognition. The markers are glued directly to the ceiling tile railing, which is eight feet above the floor.

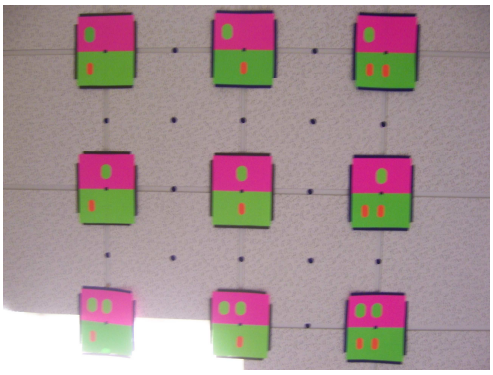


Figure 3.6: Test course coordinate grid mounted to the ceiling.

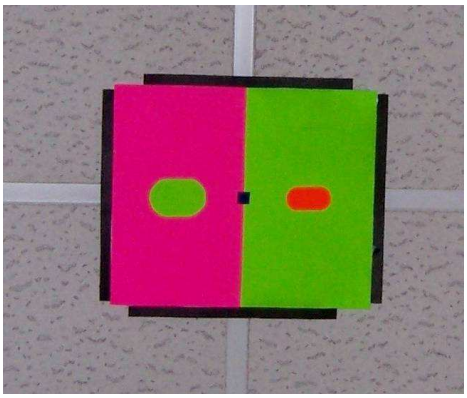


Figure 3.7: Marker for coordinate (2,2).

For the purpose of testing the new positioning system, a laser pointer was fixed to a test robot such that the laser light coincides with the robot camera center of view. The laser allows the user to place the robot at a known position within the new grid. The robot's exact position is illuminated by the laser pointer in the overhead grid.

3.3.2 New software

The new code builds upon the *userColorVisionFuncs.c* file developed by Daniel Block and implements the algorithm from Chapter 11 in [1]. The segmentation, object identification and object characteristic calculations are kept the same as the original code. The processes are duplicated since there are now two colors of interest. An additional set of definitions was created for the added functions: color definition parameters (*hue, sat, value*), measurements and dimensions for the marker template matching, error tolerances, and coordinate transformation parameters. With the exception of the scaling constant *β=pixels/tiles* [15], parameters were determined experimentally. A new set of functions was developed for the positioning algorithm. The functions were created to perform a specific task each. This makes debugging easier and keeps the functionality of the original algorithm as changes are made.

After the image is thresholded and objects are identified, the next step is template matching. A virtual template of the rectangles in the markers is built in image coordinates and is compared against the rectangles seen. The template is superimposed into each rectangle found and if the rectangle fits within the template, within some acceptable range, it is marked as a good fit and its index is recorded in an array. Criteria for a good template fit were found by pixel-counting the image and are specified in the error tolerances defined within the code. The `Couple_Rectangles()` function is then called. This function defines a proximity radius that establishes a rectangle pair. For each rectangle of *color 1* the function loops through those from *color 2*, and the one

that satisfies the proximity requirement is marked as its pair. It can be safely assumed that only one possible pair exists because the fitting process filters out any other smaller object that may be closer to the rectangle and be misclassified as a pair (i.e. dots). The array is then reorganized so the rectangle pairs are grouped within the same index in the array. Now the `Read_Dots()` function is called. This function reads the binary coordinates from a specified marker and returns the decimal values to a two-dimensional global array. For this, it makes use of the `Heading()` function. This function's purpose is to provide a heading vector for the robot's orientation and determine the MSB. Now that the markers have been identified and read, the positioning technique makes the final conversion to world frame coordinates. There are two techniques explored in this project: change of axes and barycentric coordinates. Figure 3.8 shows a flow diagram for the new algorithm. Note that the original *userColorVisionFuncs.c* algorithm is closely followed until *Step 4*.

3.3.2.1 Positioning by barycentric coordinates

The *Barycentric mode* finds the areal coordinates of the image center (robot position) with respect to the triangle formed by only the first three markers listed in the good fits array. Initial experimentation pointed to little or negligible distortion in the robot image, hence no mapping is performed. It is then correct to assume that triangle ratios in the image frame remain unchanged in the world frame and the areal coordinates computed in the former are valid for the latter. A linear algebraic problem is formed with the marker centroid coordinates and the areal coordinates. The marker centroid coordinates are

substituted with world frame coordinates on the markers and the linear problem is then solved for the world coordinates; the result is stored in global variables to be used by the controller in place. Naturally, the fundamental requirement when operating in *Barycentric Mode* is that at least three markers have to be successfully seen and identified. This may prove to be a restriction when navigating in the border areas of the course as markers may become scarce.

3.3.2.2 Change of axes

The *Change of Axes Mode* applies a linear transformation to the image frame coordinates and changes the axes to world frame coordinates. The rotation matrix is constructed with the orientation angle provided by the `Heading()` function and the scaling factor β – known *a priori*. This mode's versatility over *Barycentric Mode* is evident when it is noted that only one successful marker recognition is needed.

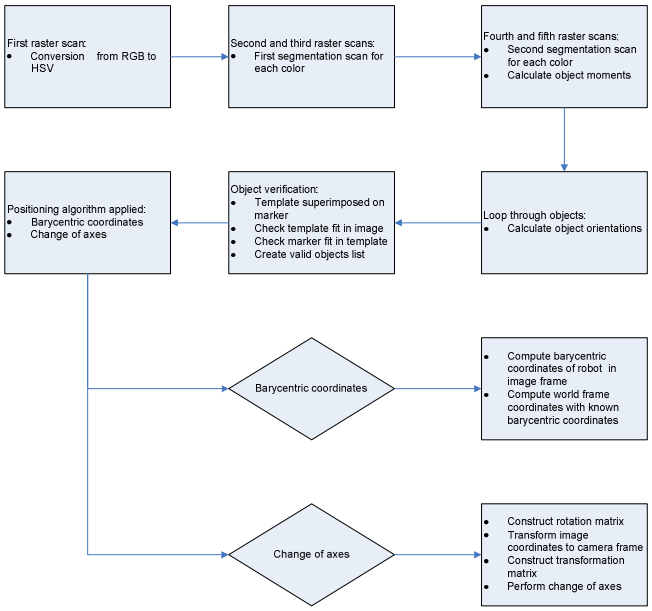


Figure 3.8: Positioning algorithm flow diagram.

3.3.2.3 Functions created

A list of the functions developed for this project along with their descriptions follows. Each function description includes the input arguments that the function requires, its output result if any, and flow diagrams to display the series of tasks performed.

`Couple_Rectangles()`

argument: none

return value: none

The function scans through the global array that contains all recognized markers from both colors. The array includes a counter of figures found for each color. The lowest value of the counters determines the maximum number of pairs that can exist; this number is rewritten as the new value for both counters and will be used by the algorithm to determine which positioning method is to be applied. The search starts from the first object found in *color 1*. For each object in *color 1*, all objects in *color 2* are searched to find the one whose centroid falls within a predetermined radius. The *color 2* pairs are rewritten in the global array such that all color pairs have the same index. Figure 3.9 shows a flow diagram of the operation performed by the function.

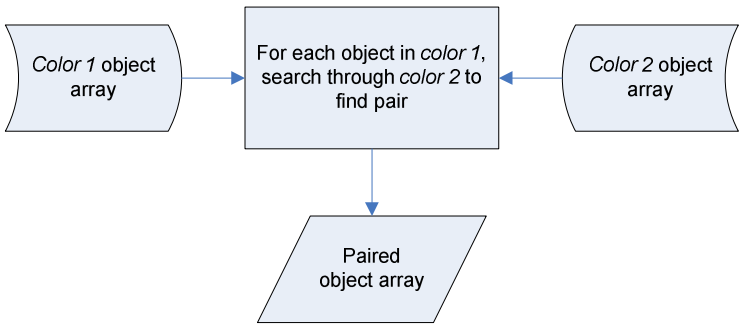


Figure 3.9: *Couple_Rectangles( )* function flow diagram.

`Heading()`

argument: object number

return value: integer [0, 3]

Two orientation measurements are computed in the `Heading()` function. The function first constructs a heading vector perpendicular to the line that forms at the union of the green and red regions of the marker. A heading angle is calculated by computing `atan2` of the vector. The second output signals to the general cardinal direction that the robot is facing. This is done by following the definition that green marks West. With the cardinal direction known, the MSB can now be identified. This is explained in the next function. Figure 3.10 shows the operation of `Heading()`.

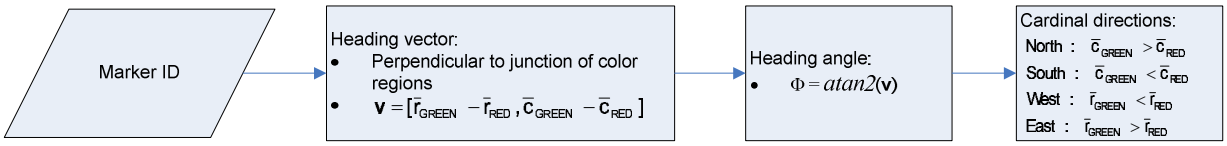


Figure 3.10: *Heading( )* function flow diagram.

`Read_Dots()`

argument: marker number

return value: none

The function specifies which marker to read in the good fits array. For each rectangle inside the marker, the function scans the length of the rectangle, along its major axis, and records how many dots are found. The function then compares the centroid of each dot on the list with predefined slots in a template. Each slot represents a bit of a 3-bit binary number. When all dots have been localized to within their respective bit slot, an array is created with the binary numbers read from the marker. The function calls the

`Heading()` subroutine and determines the MSB of the binary number from the cardinal direction to which the robot is pointing. By definition, the binary numbers are read from North to South. With the heading known, the subroutine chooses the correct bit order to read. The binary numbers are then converted to decimal base and returned to a global array. Figure 3.11 depicts the order of events within `Read_Dots()`.

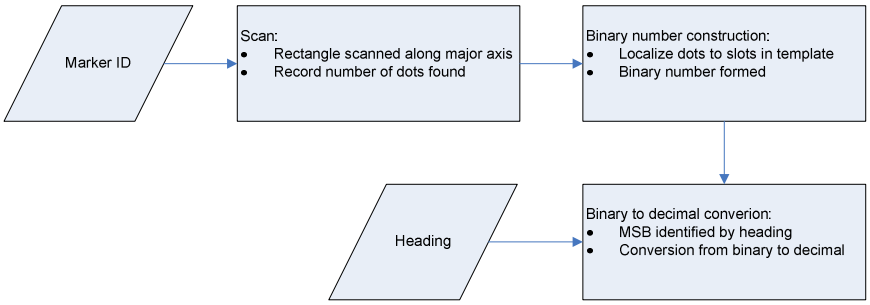


Figure 3.11: ***Read_Dots( )*** **function flow diagram.**

`Barycentric()`

argument: none

return value: integer [0, 1]

The function solves Eq. (2.7) from Section 2.8. The **R** matrix is constructed with the image coordinates of the marker centroids. The value of λ_3 is calculated from Eq. (2.5), shown in Fig. 3.12. The areal coordinates are stored in a global array and an integer is returned: 0 if the image center (robot position) is inside or on the edge of the triangle, or 1 otherwise.

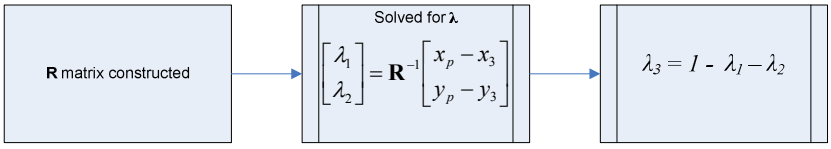


Figure 3.12: ***Barycentric( )*** **function flow diagram.**

4. EXPERIMENT AND RESULTS

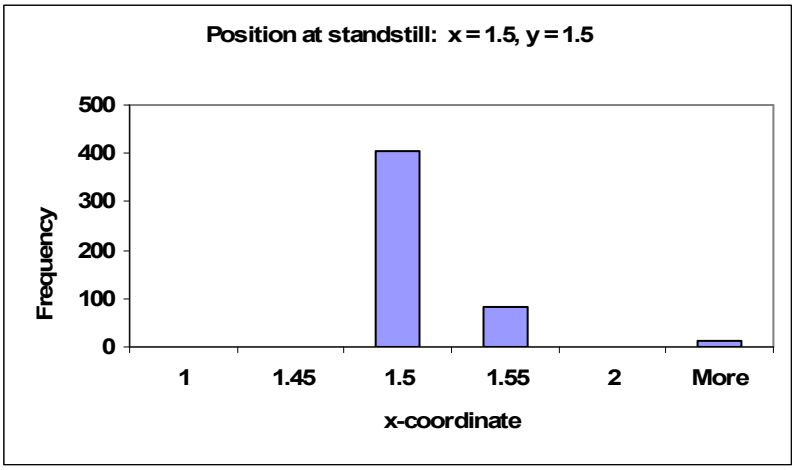
Two experiments were performed to test the validity and performance of the positioning system. Each algorithm mode was tested. Both experiments were performed with the same robot and in the same course.

Nine markers were constructed and arranged on the ceiling tiles to form a square, three-by-three grid to serve as the test course. Small black dots were added between markers as a visual aid. The robot was fitted with a laser pointer, aimed to the ceiling and marking the center of the camera image. The vehicle was manually positioned exactly below position markers. At standstill, position and orientation data was recorded for 500 consecutive data points and downloaded to MATLAB via serial connection.

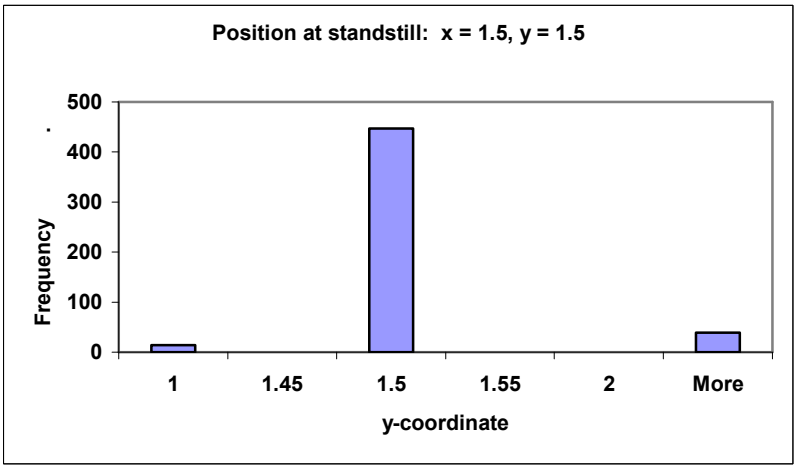
Only static measurements were performed and only position data was considered for analysis in this thesis. There was no method in place for verification of true heading, so orientation data points were not analyzed. Histograms for orientation data are included in Appendix A and Appendix B for *Change of Axes Mode* and *Barycentric Mode*, respectively. The robot was placed under the same set of coordinates in the grid and positioning was calculated under both operating modes. A limited set of coordinates were used on both experiments, limited by the *Barycentric Mode*'s performance. For the *Barycentric Mode* test, the coordinates were chosen such that the robot camera had the clearest view possible of three markers. This set is the smaller of the two modes; as a result, the same set for *Change of Axes Mode* was used for comparison.

4.1 Barycentric Mode Results

This section presents the data gathered during the experiment. Static measurements for position using *Baryentric Mode* of operation are displayed in histogram form in Figs. 4.1-4.5. Unevenly distributed bins were used to emphasize the interval of ±0.05 units from the exact position.

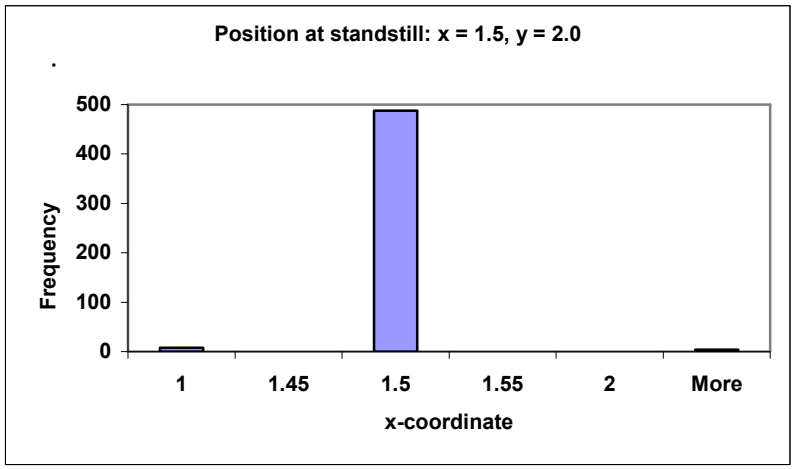


(a)

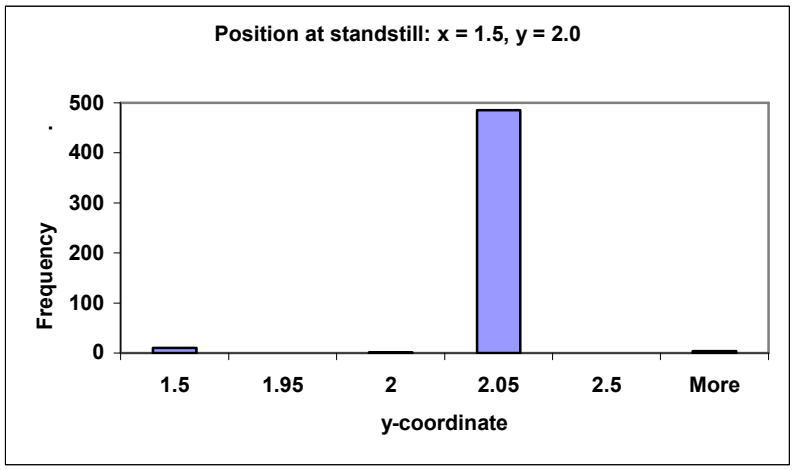


(b)

Figure 4.1: Histograms for coordinates (1.5, 1.5) at standstill using *Barycentric Mode*. Total of 500 data points recorded. (a) shows 98% of data points lie within ±0.05 of the true value, (b) shows 89% of points lie within ±0.05 of true value.

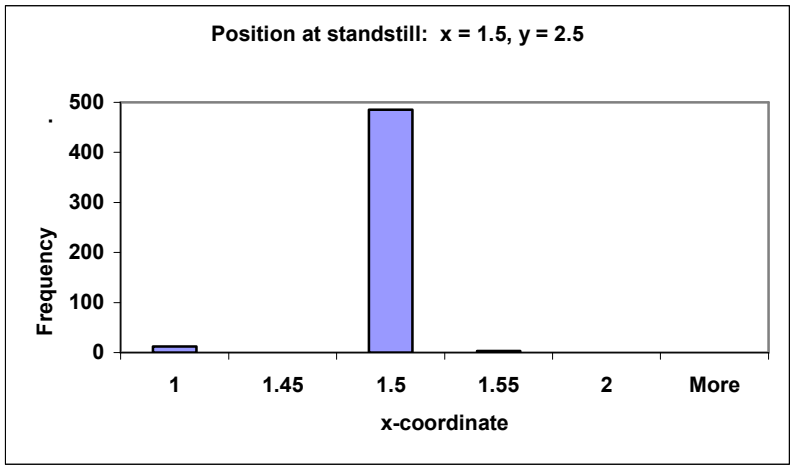


(a)

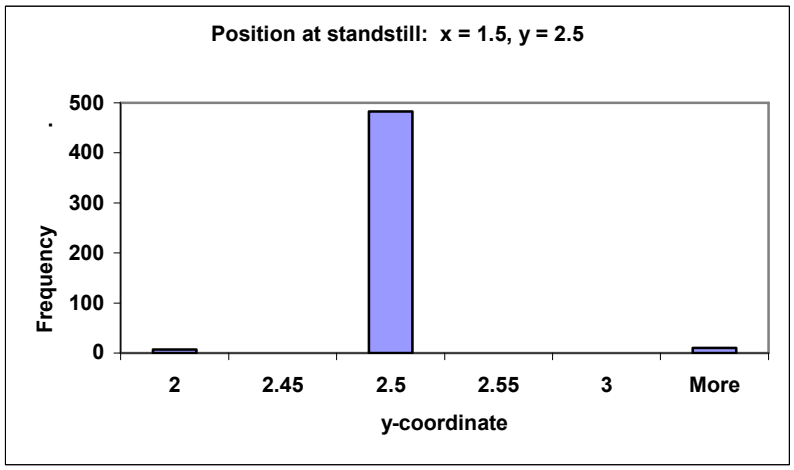


(b)

Figure 4.2: Histograms for coordinates (1.5, 2.0) at standstill using *Barycentric Mode*. Total of 500 data points recorded. (a) shows 98% of data points lie within ±0.05 of the true value, (b) shows 97% of points lie within ±0.05 of true value.

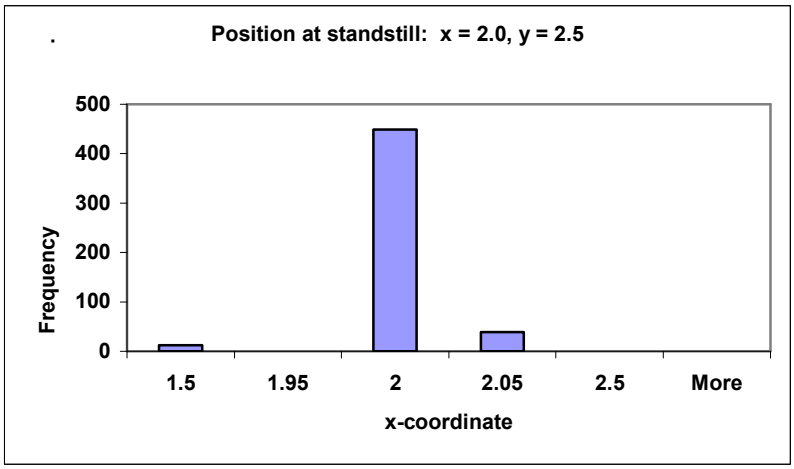


(a)

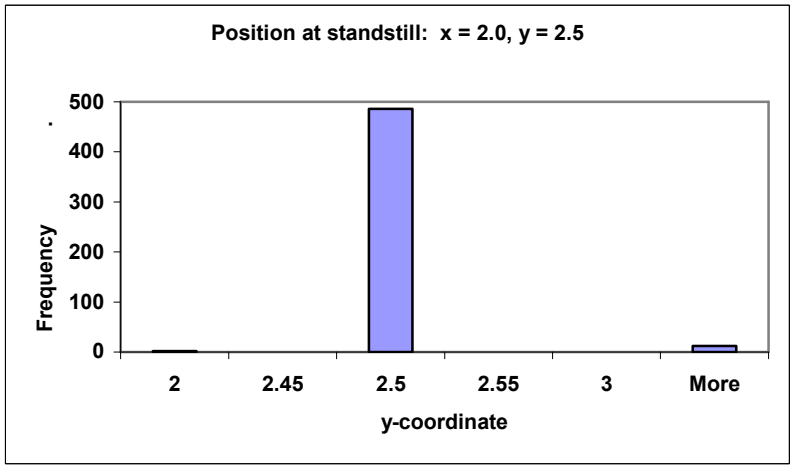


(b)

Figure 4.3: Histograms for coordinates (1.5, 2.5) at standstill using *Barycentric Mode*. Total of 500 data points recorded. (a) shows 98% of data points lie within ±0.05 of the true value, (b) shows 97% of points lie within ±0.05 of true value.

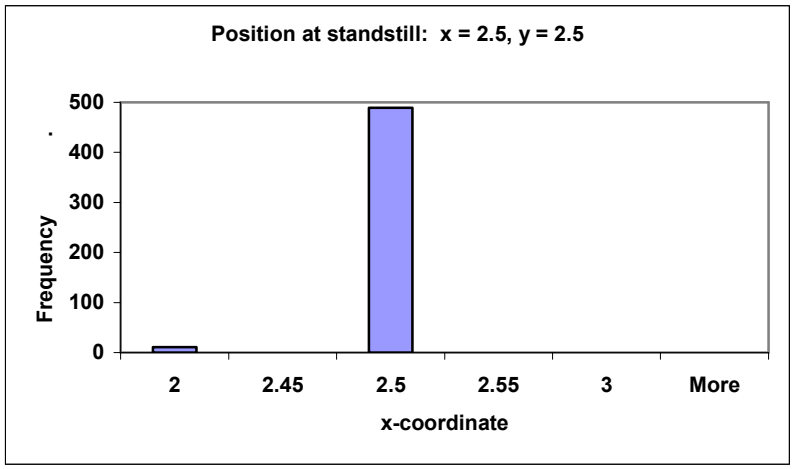


(a)

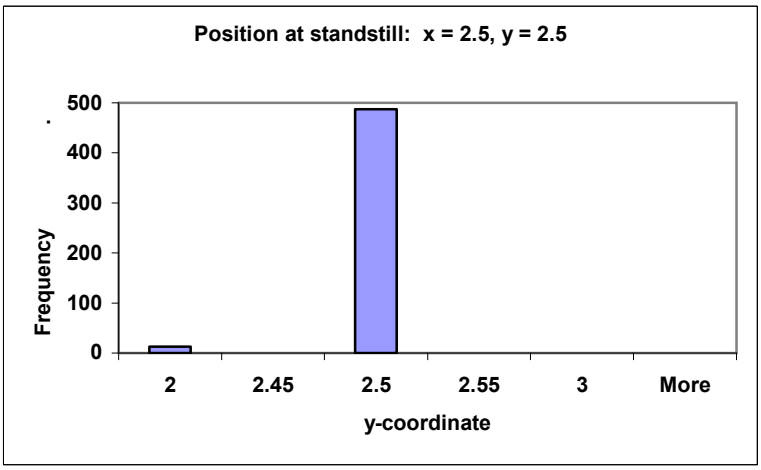


(b)

Figure 4.4: Histograms for coordinates (2.0, 2.5) at standstill using *Barycentric Mode*. Total of 500 data points recorded. (a) shows 98% of data points lie within ±0.05 of the true value, (b) shows 97% of points lie within ±0.05 of true value.



(a)

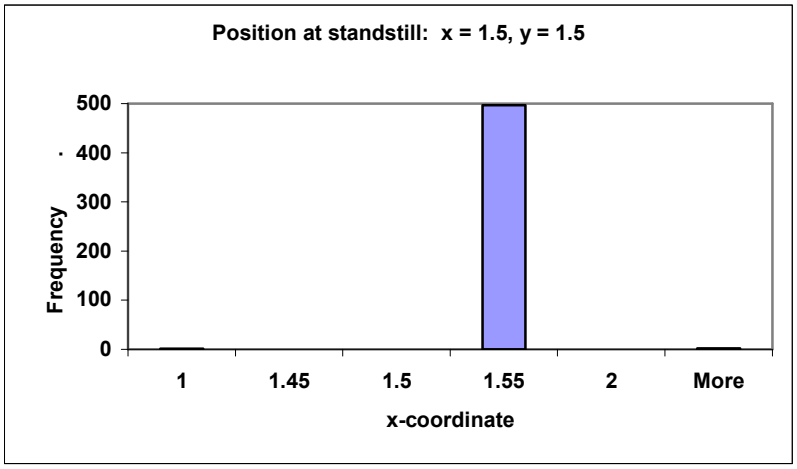


(b)

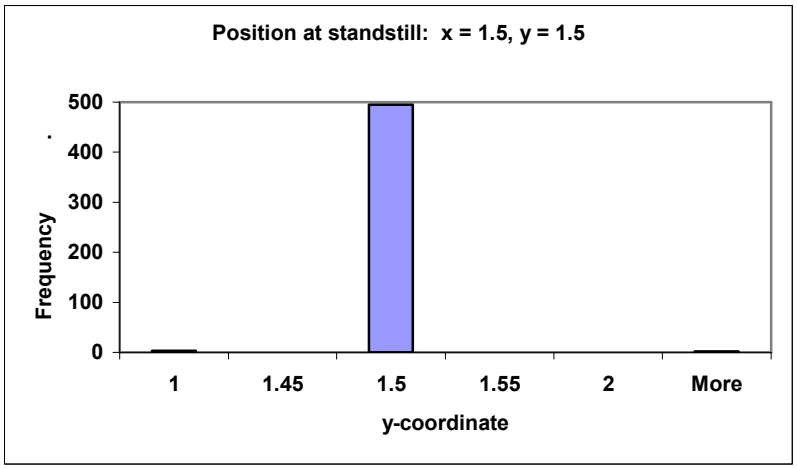
Figure 4.5: Histograms for coordinates (2.5, 2.5) at standstill using *Barycentric Mode*. Total of 500 data points recorded. (a) shows 98% of data points lie within ±0.05 of the true value, (b) shows 97% of points lie within ±0.05 of true value.

4.2 Change of Axes Mode Results

This section presents the data gathered during the experiment. Static measurements for position using *Change of Axes Mode* of operation are displayed in histogram form in Figs. 4.6-4.10. Unevenly distributed bins were used to emphasize the interval of ±0.05 units around true position and the percentage of the points in the interval was calculated.

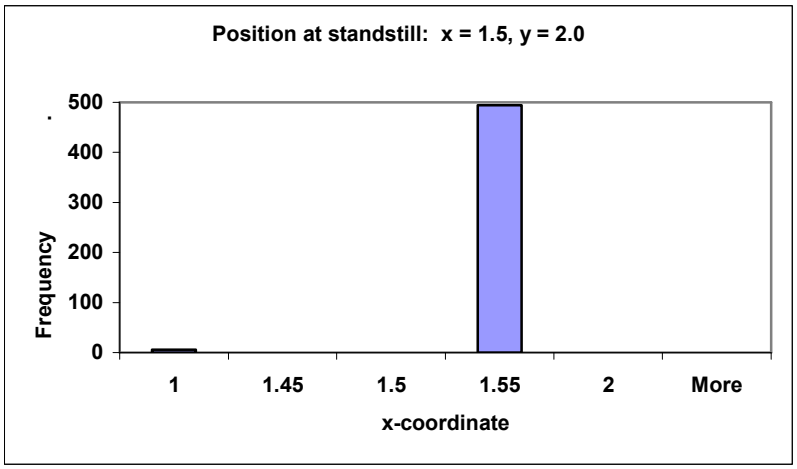


(a)

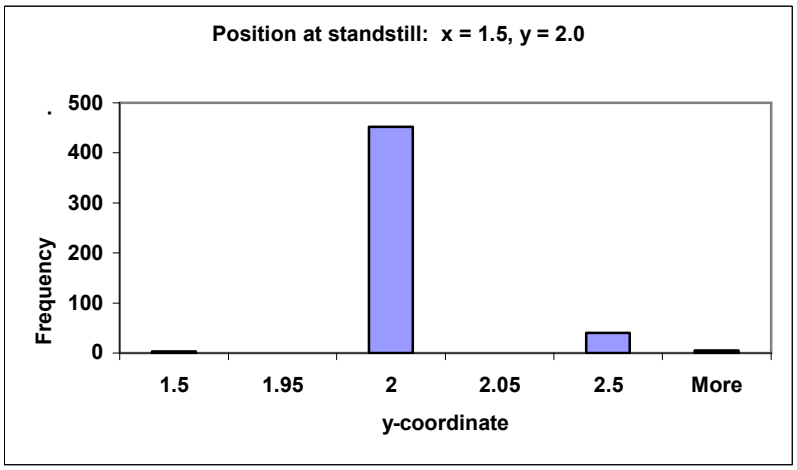


(b)

Figure 4.6: Histograms for coordinates (1.5, 1.5) at standstill using *Change of Axes Mode*. Total of 500 data points recorded. (a) shows 99% of data points lie within ±0.05 of the true value, (b) shows 99% of points lie within ±0.05 of true value.

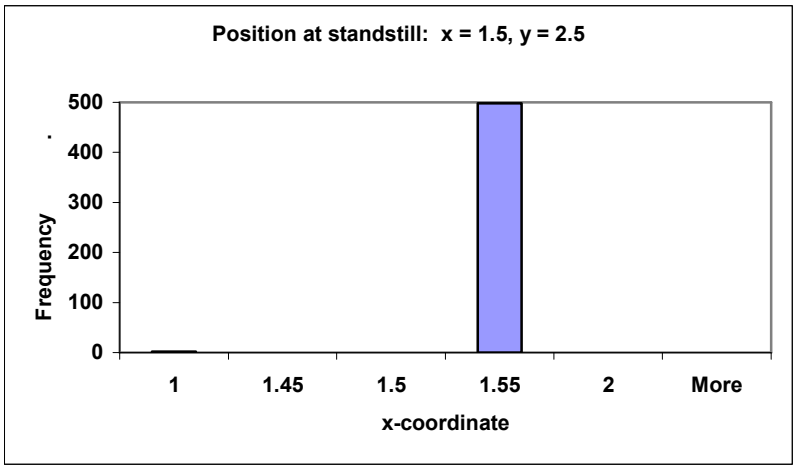


(a)

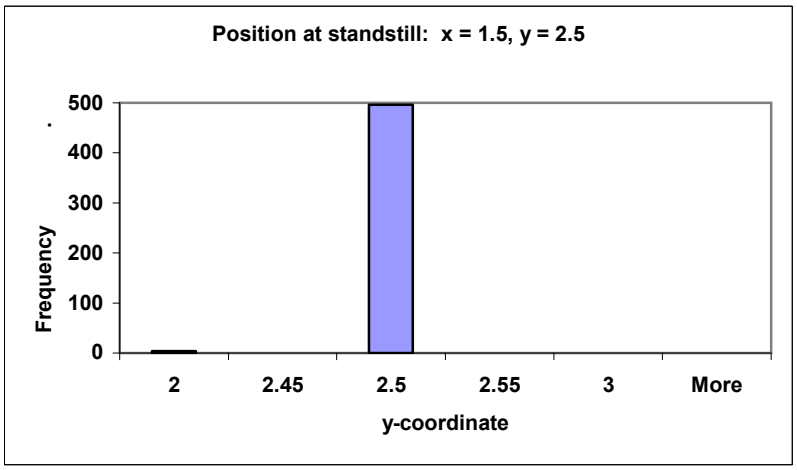


(b)

Figure 4.7: Histograms for coordinates (1.5, 2.0) at standstill using *Change of Axes Mode*. Total of 500 data points recorded. (a) shows 99% of data points lie within ±0.05 of the true value, (b) shows 90% of points lie within ±0.05 of true value.

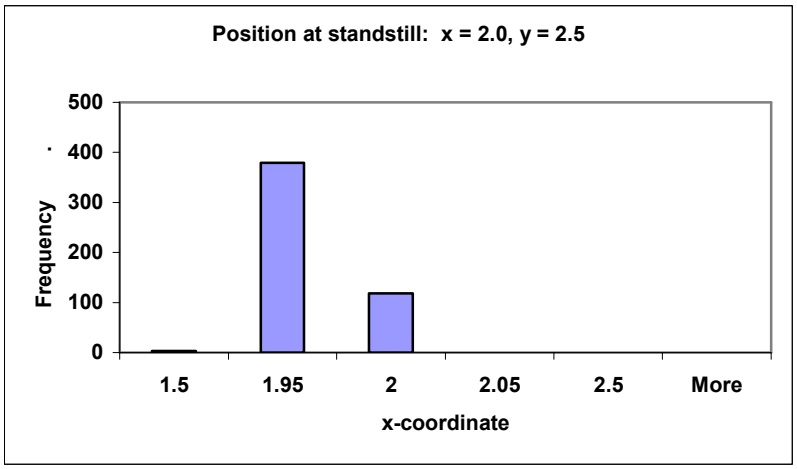


(a)

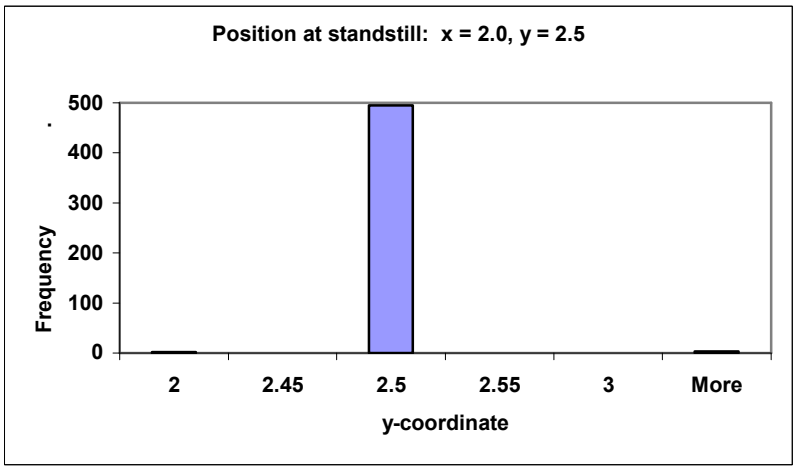


(b)

Figure 4.8: Histograms for coordinates (1.5, 2.5) at standstill using *Change of Axes Mode*. Total of 500 data points recorded. (a) shows 100% of data points lie within ±0.05 of the true value, (b) shows 99% of points lie within ±0.05 of true value.

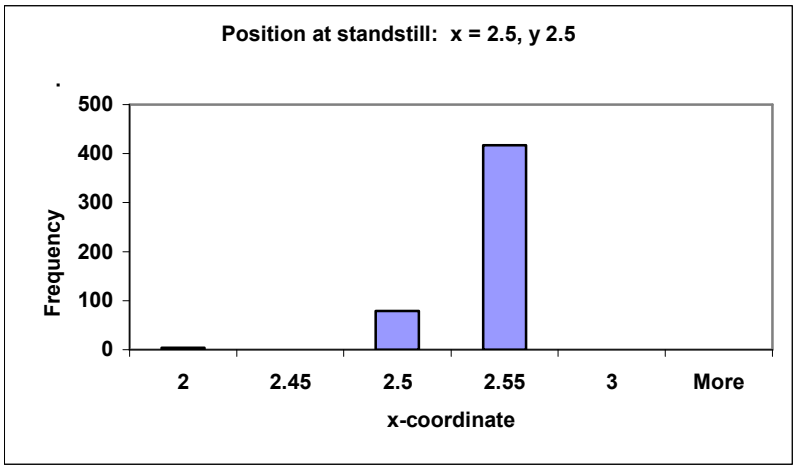


(a)

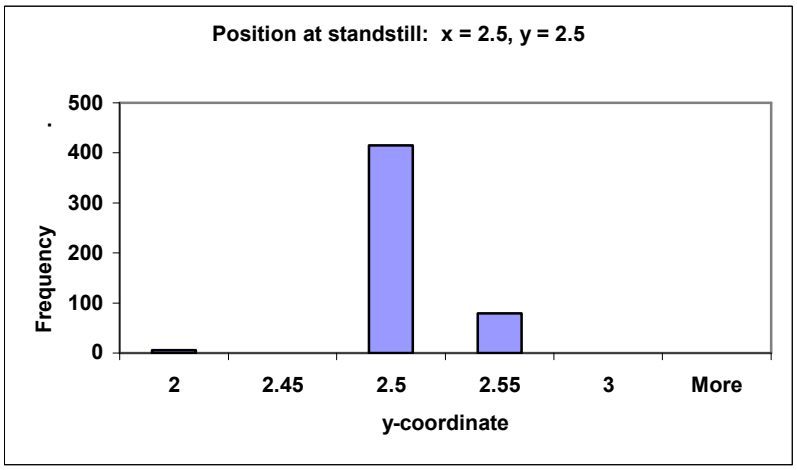


(b)

Figure 4.9: Histograms for coordinates (2.0, 2.5) at standstill using *Change of Axes Mode*. Total of 500 data points recorded. (a) shows 99% of data points lie within ±0.05 of the true value, (b) shows 99% of points lie within ±0.05 of true value.



(a)



(b)

Figure 4.10: Histograms for coordinates (2.5, 2.5) at standstill using *Change of Axes Mode*. Total of 500 data points recorded. (a) shows 99% of data points lie within ±0.05 of the true value, (b) shows 99% of points lie within ±0.05 of true value.

4.3 Discussion

An interval of ±0.05 units was chosen as a tolerance criterion to measure performance. Both modes display a fairly constant positioning performance of 90% or better, without any noticeable difference between them. It is important to say that some deviations from these tolerances were encountered, but these appeared sporadically and were difficult to replicate. Also, it was observed that a semi-periodic set of outliers appeared in the measurements. The outliers seem to be trash data from memory. Almost all sets of data points exhibited this behavior. It is suspected that the algorithm takes too much time to execute. Since the DSP/BIOS platform works under a priority scheme, the positioning algorithm may not be able to finish before the DSP interrupts the function to execute another one of higher hierarchy. This hypothesis is based on the fact that the algorithm takes 400 ms to execute. This is the main suspected reason, but further work may be needed to investigate this behavior.

Although the robot remained at standstill through all measurements, data points show variations. These are caused by varying light conditions on the markers that in turn cause variations in the object characteristics in the digital image. The course is placed on a working laboratory area where lighting conditions cannot be controlled. In an effort to achieve a more even light distribution, improvised diffusers were placed over the lighting fixtures. This helps to some extent but the problem still persists. Also, the course is currently limited to the space between lighting fixtures. Positions on or near the edge of a fixture are hard to identify because the direct light causes the camera's auto-exposure

compensation to adjust the gain. These changes are too large for the algorithm to accommodate and color definitions are no longer valid under these conditions. For this reason the course is limited to a three by three square grid and available positions to test were limited.

Orientation data is presented in Appendixes A and B. No true heading measurement was available to compare data validity. However, measurements were recorded and statistically graphed. From observation it can be noted that orientation calculations have a lower level of performance. This should not be taken as a fact, and further testing is needed.

5. CONCLUSIONS AND RECOMMENDATIONS

A new positioning system has been proposed for the B16 Robotics Laboratory testbed. The system is vision based and distributed. Each agent is responsible for solving its individual positioning problem. The new code developed adds to the existing code that implements the algorithm in Chapter 11 of [1]. The new course layout is marked by templates on ceiling tiles that consist of two joined rectangles with dots inside. The rectangle color tells the algorithm which axis the coordinates inside it belong to: red for x, green for y. Orientation information is provided by a vector created from red to green and this direction is defined as West. Coordinates are displayed inside the markers in binary form; 3 bits are currently in use. The algorithm implements detection and identification techniques for the markers, following a template matching approach. Two modes of operation were created, *Barycentric* and *Change of Axes*.

The new system is able to calculate position using the two methods and 90% or more of data points in each set recorded lie within 0.05 units of true position. Heading was also calculated, but further testing is needed to asses its performance.

A full course test was not possible because the space is limited to the areas between lamp fixtures, and areas surrounding the web cameras have to be avoided as well. The algorithm's time of execution (TOE) was measured to be 400 ms. This supports the idea that the entire function cycle is not being executed and yields inconsistencies on data points.

Change of Axes Mode proved to be reliable because of the other mode's requirement to identify at least three markers successfully. Lighting conditions have a considerable impact on the algorithm's performance, but color detection performed fairly well on uniformly lighted regions.

The concept for the new positioning system was validated. The system achieved partial functionality and acceptable performance. However, it has not yet reached the potential necessary to replace the current system. The problem of sharing position data among robots for multi-agent operations still needs to be addressed. There is much room for improvement in the code execution. Changes in priority and order of execution may streamline the algorithm and decrease its TOE. Adjusting image size or camera frame rate may free more time for the DSP to process the images completely. Also, additional techniques may be applied for template matching. Two subroutines that searched for template fits and adjusted the object characteristics accordingly are left as further work to improve performance. A simple filtering process may also be applied for smoothing out the data. Other techniques may be explored for color segmentation or position calculation. When three or more markers are identified, other positioning mechanisms may be applied that take advantage of all available data. However, care must be taken with TOE since the DSP must be allowed enough processing time to execute other tasks.

REFERENCES

[1] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. New York, NY: Wiley, 2005.

[2] A. C. Frery, J. Gomes, and L. Velho, *Image Processing for Computer Graphics and Vision*. New York, NY: Springer, 2008.

[3] J. Vince, *Mathematics for Computer Graphics*. New York, NY: Springer, 2005.

[4] N. Michael, J. Fink, and V. Kumar, "Experimental testbed for large multirobot teams," *IEEE Robotics and Automation Magazine*, vol. 15, no. 1, pp. 53-61, March 2008.

[5] L. Cremean, W. B. Dunbar, D. van Gogh, J. Hickey, E. Klavins, J. Meltzer, and R. M. Murray, "The Caltech multi-vehicle wireless testbed," in *Proceedings of the 41st IEEE Conference on Decision and Control*, 2002, pp. 86-88.

[6] E. King, Y. Kuwata, M. Alighanbari, L. Bertuccelli, and J. How, "Coordination and control experiments on a multi-vehicle testbed," in *Proceedings of the American Control Conference*, 2004, pp. 5315-5320.

[7] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE Control Systems Magazine*, vol. 28, no. 2, pp. 51-64, April 2008.

[8] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, and C. J. Tomlin, "The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC)," in *The 23rd Digital Avionics Systems Conference*, 2004, pp. 12.E.4-1 – 12.E.4-10.

[9] H. J. Kim and D. H. Shim, "A flight control system for aerial robots: Algorithms and experiments," *Control Engineering Practice,* vol. 11, no. 12, pp. 1389-1400, December 2003.

[10] S. Hoyt, S. McKennoch, and L. G. Bushnell, "An autonomous multi-agent testbed using infrared wireless communication and localization," University of Washington, Seattle, Washington, Tech. Rep. UWEETR-2005-0005, 2005.

[11] D. Cruz, J. McClintock, B. Perteet, O. A. A. Orqueda, Yuan Cao, and R. Fierro, "Decentralized cooperative control - A multivehicle platform for research in networked embedded systems," *IEEE Control Systems Magazine*, vol. 27, no. 3, pp. 58-78, June 2007.

[12] A. Stubbs, V. Vladimerou, A. T. Fulford, D. King, J. Strick, and G. E. Dullerud, "Multivehicle systems control over networks: A hovercraft testbed for networked

and decentralized control," *IEEE Control Systems Magazine*, vol. 26, no. 3, pp. 56-69, June 2006.

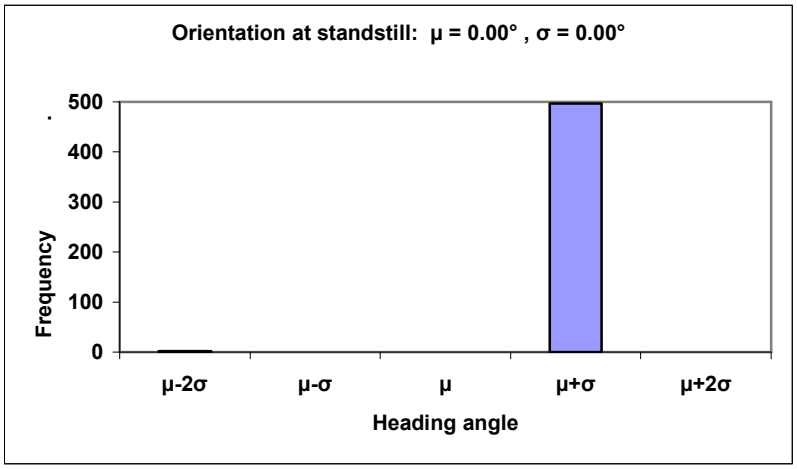
[13] Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, "GE 423 - Laboratory handouts, notes, & support files," September 2009. [Online]. Available: http://coecsl.ece.uiuc.edu/ge423/handouts.htm.

[14] T. R. Brdar, "Robotic testbed development and time delay compensation," M.S. thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 2008.

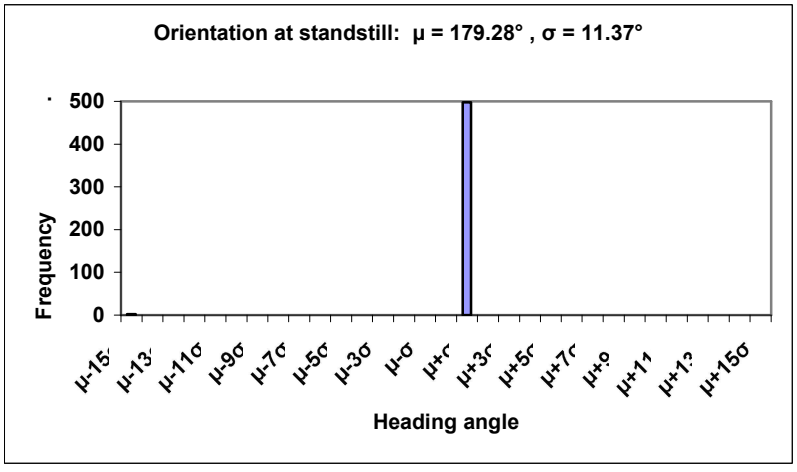
[15] J. K. Holm, *Introduction to Robotics Lab Manual*, Urbana, IL, University of Illinois at Urbana-Champaign, 2007.

APPENDIX A: ORIENTATION DATA FOR CHANGE OF AXES MODE

This appendix presents the orientation data gathered during the experiment. Static measurements using *Change of Axes Mode* of operation are displayed in histogram form in Figs. A.1-A.5. Percentage of data points falling within one standard deviation of mean value was calculated.

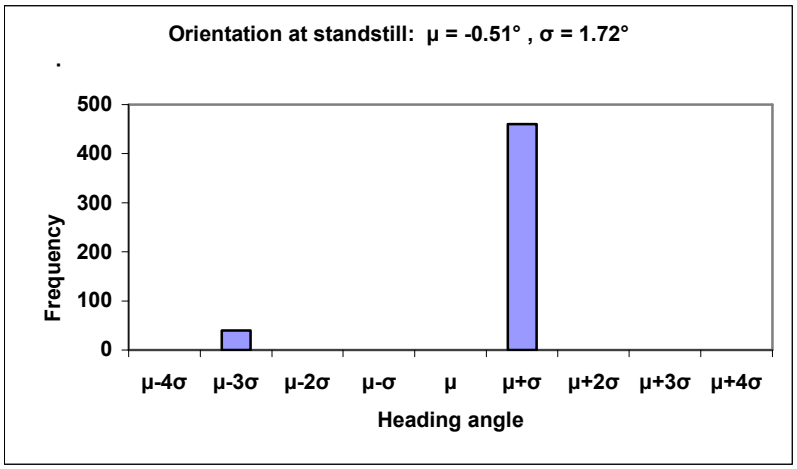


(a)

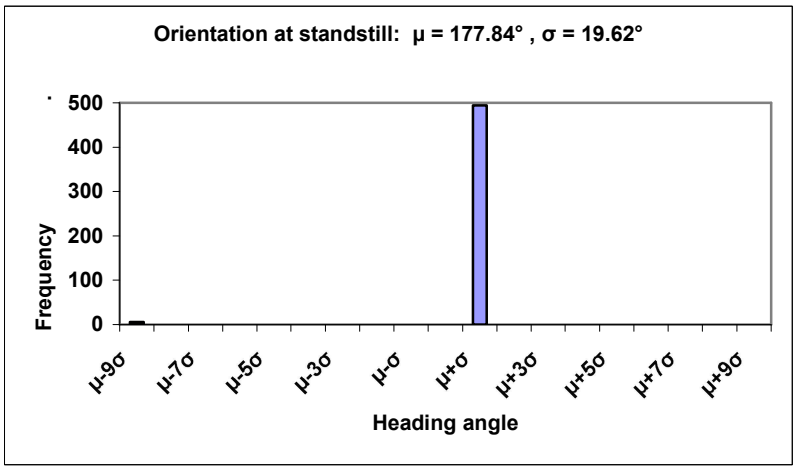


(b)

Figure A.1: Histograms for coordinates (1.5, 1.5) at standstill using *Change of Axes Mode*. Total of 500 data points recorded. (a) shows 99.4% of data points lie within ±1σ of the mean value, (b) shows 99.6% of points lie within ±1σ of mean value.

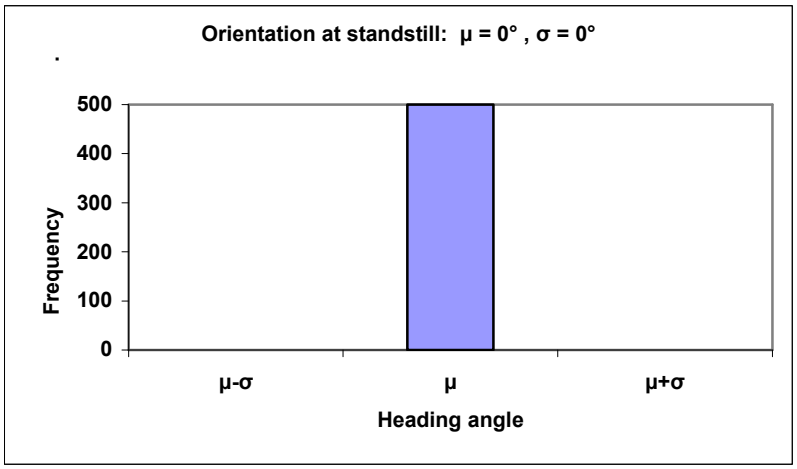


(a)

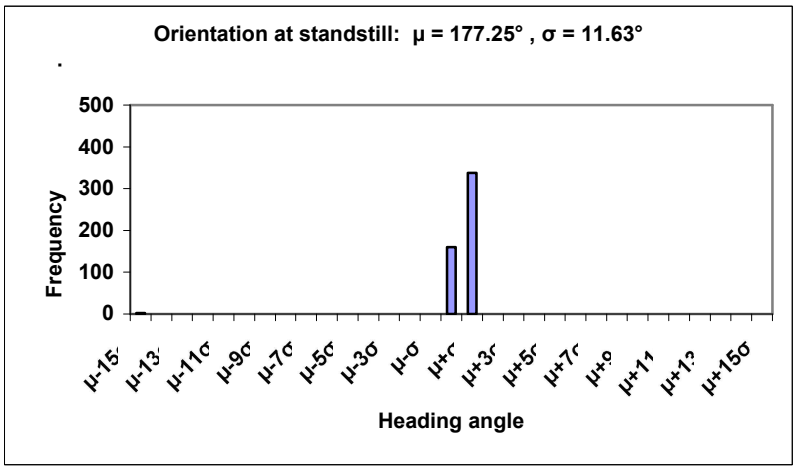


(b)

Figure A.2: Histograms for coordinates (1.5, 2.0) at standstill using *Change of Axes Mode*. Total of 500 data points recorded. (a) shows 92.0% of data points lie within ±1σ of the mean value, (b) shows 98.8% of points lie within ±1σ of mean value.

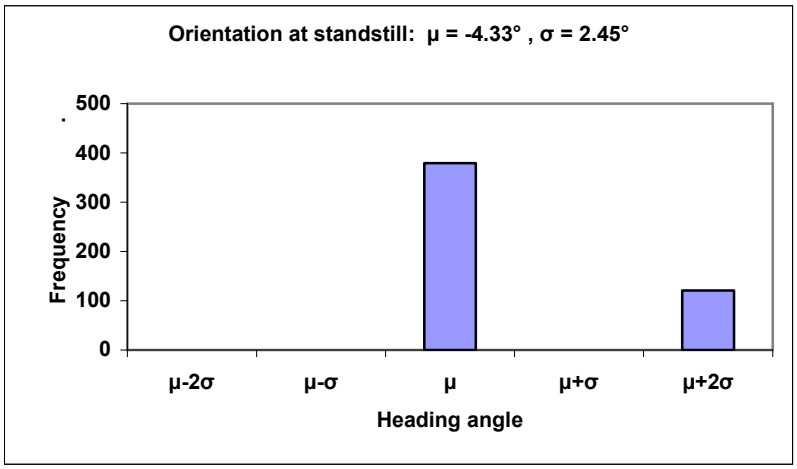


(a)

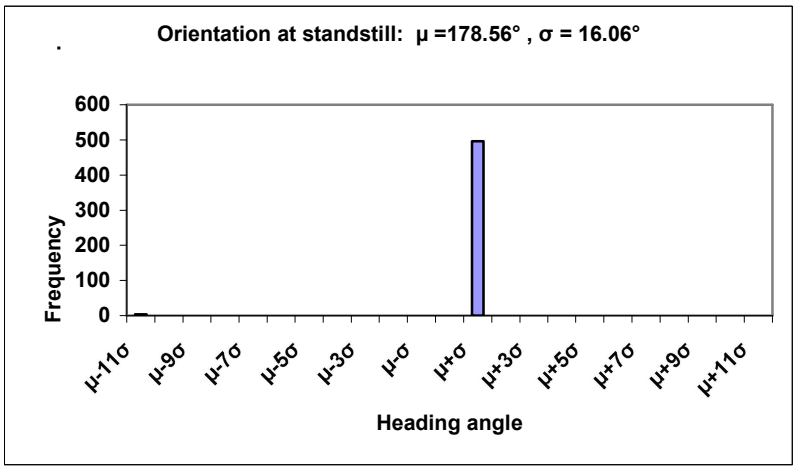


(b)

Figure A.3: Histograms for coordinates (1.5, 2.5) at standstill using *Change of Axes Mode*. Total of 500 data points recorded. (a) shows 100% of data points lie within ±1σ of the mean value, (b) shows 99.6% of points lie within ±1σ of mean value.

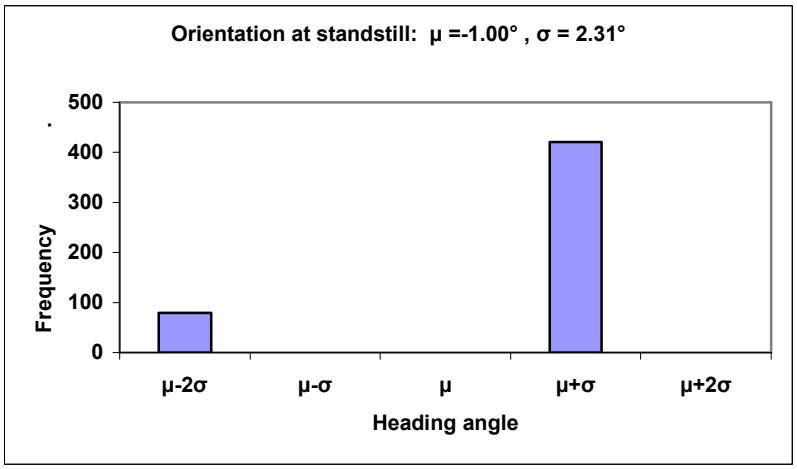


(a)

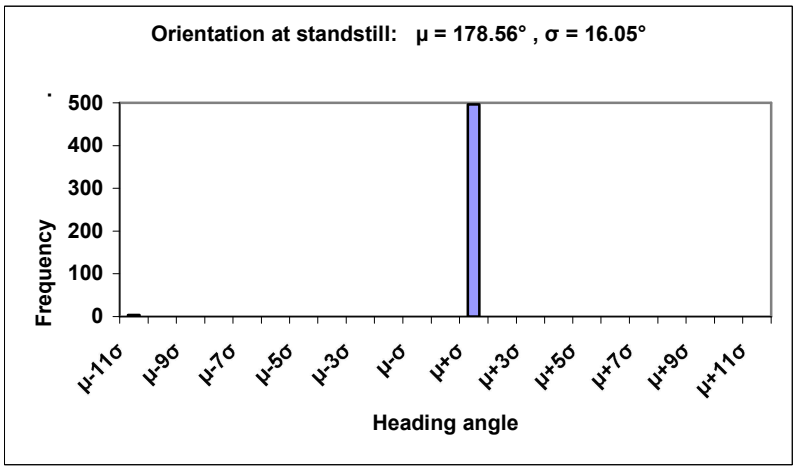


(b)

Figure A.4: Histograms for coordinates (2.0, 2.5) at standstill using *Change of Axes Mode*. Total of 500 data points recorded. (a) shows 75.8% of data points lie within ±1σ of the mean value, (b) shows 99.2% of points lie within ±1σ of mean value.



(a)



(b)

Figure A.5: Histograms for coordinates (2.5, 2.5) at standstill using *Change of Axes Mode*. Total of 500 data points recorded. (a) shows 84.2% of data points lie within ±1σ of the mean value, (b) shows 99.2% of points lie within ±1σ of mean value.

APPENDIX B: ORIENTATION DATA FOR BARYCENTRIC MODE

This appendix presents the orientation data gathered during the experiment. Static measurements using *Barycentric Mode* of operation are displayed in histogram form in Figs. B.1-B.5. Percentage of data points falling within one standard deviation of mean value was calculated.

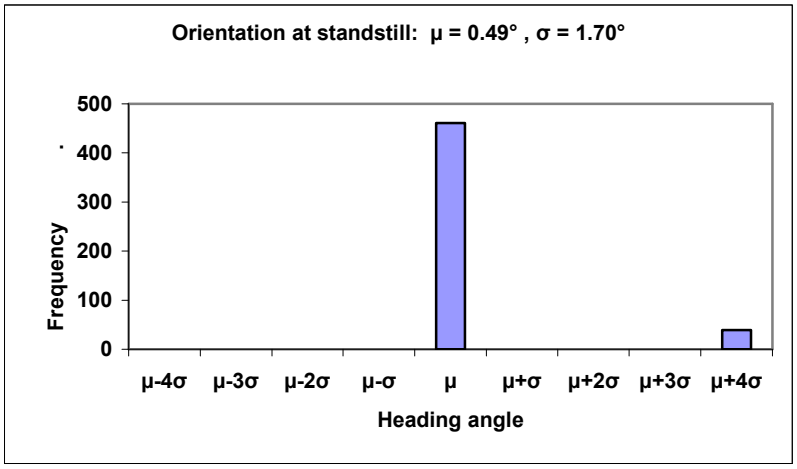


Figure B.1: Histogram for coordinates (1.5, 1.5) at standstill using *Barycentric Mode*. Total of 500 data points recorded. Figure shows 92.2% of data points lie within ±1σ of the mean value.

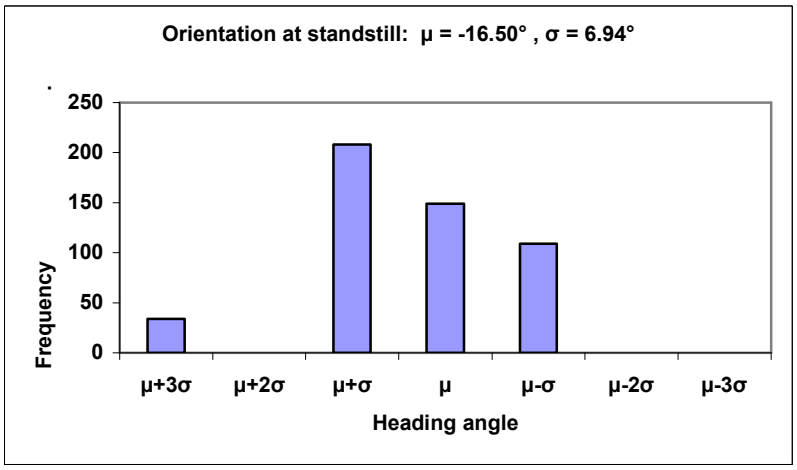


Figure B.2: Histogram for coordinates (1.5, 2.0) at standstill using *Barycentric Mode*. Total of 500 data points recorded. Figure shows 93.2% of data points lie within ±1σ of the mean value.

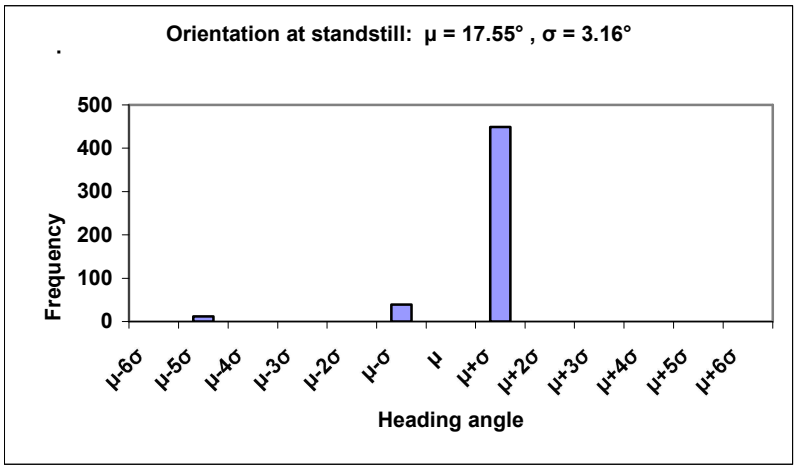


Figure B.3: Histogram for coordinates (1.5, 2.5) at standstill using *Barycentric Mode*. Total of 500 data points recorded. Figure shows 97.6% of data points lie within ±1σ of the mean value.

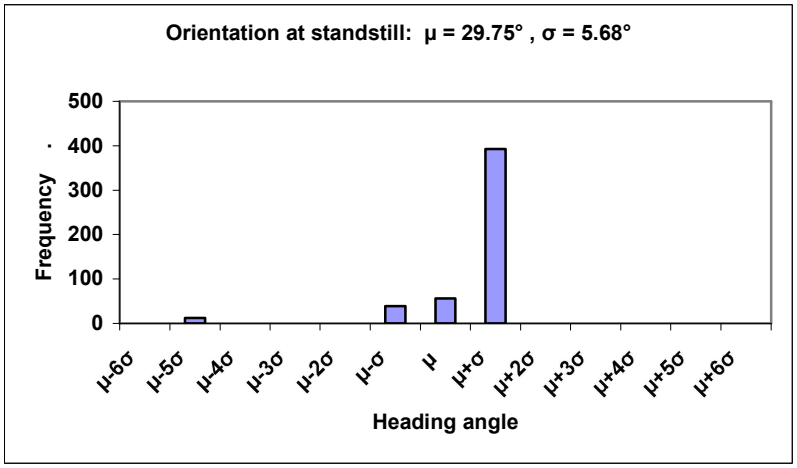


Figure B.4: Histogram for coordinates (2.0, 2.5) at standstill using *Barycentric Mode*. Total of 500 data points recorded. Figure shows 97.6% of data points lie within ±1σ of the mean value.

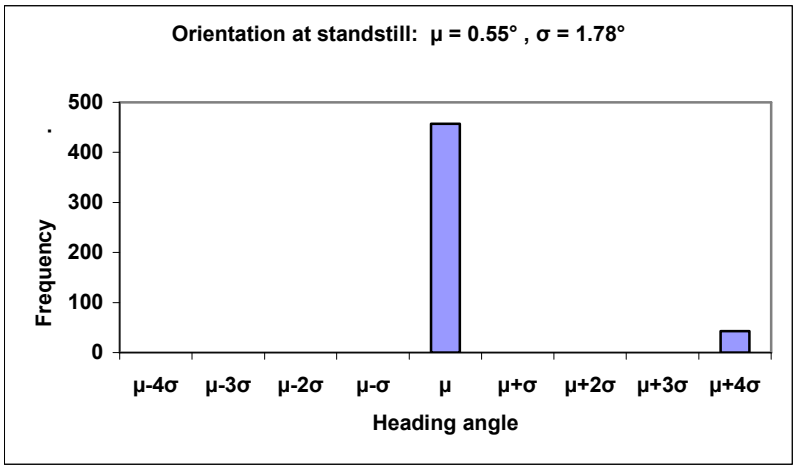


Figure B.5: Histogram for coordinates (2.5, 2.5) at standstill using *Barycentric Mode*. Total of 500 data points recorded. Figure shows 91.4% of data points lie within ±1σ of the mean value.