12-2006

A robotics testbed : the design & implementation with applications.

Travis Alan Riggs
University of Louisville

A ROBOTICS TESTBED: THE DESIGN & IMPLEMENTATION WITH APPLICATIONS

By

Travis Alan Riggs
Bachelor of Science in Electrical Engineering, 2005
Speed School of Engineering
University of Louisville

A Thesis
Submitted to the Faculty of the
University of Louisville
Speed School of Engineering
as Partial Fulfillment of the Requirements
for the Professional Degree of

MASTER OF ENGINEERING

Department of Electrical Engineering

December 2006

A ROBOTICS TESTBED: THE DESIGN & IMPLEMENTATION WITH APPLICATIONS

Submitted by

Travis Alan Riggs

A Thesis Approved on

_____

(Date)

by the Following Reading and Examination Committee:

_____

Tamer Inanc, Thesis Director

_____

Joseph D. Cole

_____

Andrea Kelecy

ABSTRACT

In order to study the movements of autonomous mobile robots, a tool is needed to quantify those movements. A testbed is an apparatus that provides a designated space for multiple mobile robots while tracking their position in real-time. That tracking information can also be communicated to the robots themselves to serve as closed-loop feedback. With this tool, many techniques can be developed and validated through various control experiments. A design and implementation of a testbed is presented here. The testbed is analyzed for its performance and several applications are presented to demonstrate its usefulness.

ACKNOWLEDGMENTS

The author gives all praise and honor to God for the ability to complete this project. The author would like to thank the thesis advisor, Tamer Inanc, for his direction. He would also like to thank his lovely wife, Jaime, for her continued support through this undertaking.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I
INTRODUCTION

Imagine a robot entering a collapsed building to search for survivors. Picture a machine assisting an elderly woman at the supermarket, or helping her cross the street. Envision a robot inspecting passengers at a border crossing in a war-torn country. Think of autonomous vehicles roaming the surface of nearby planets in search of minerals and resources.

From the dull to the dangerous, robots have the potential to impact our world in a promising way. Machines could perform simple, mundane tasks, freeing people to perform higher level work. Or they could protect lives.

These are just some of the applications of autonomous mobile robotics. Each presents its own challenges and needs, melding a broad range of disciplines from path planning to sensor fusion. To realize these dreams, more work must be done. In order to study some of these areas, a tool must be developed to validate new methods for autonomous robots. For the area of control systems, that tool is a testbed.

Testbeds have been used for some time to study various aspects of robotic movements [1]-[7]. A testbed typically consists of a sizable area for the robot or robots to move around and is completed by some system to measure their movement. Traditionally, this system is vision-based. Video cameras are mounted above and face downward to view the testbed floor. Each robot is tracked from the video feed and the pose information is then communicated to the robots below. In this way, the cameras act like a local global positioning satellite (GPS) system. Equipped with this tool, many new techniques might be developed and validated through different open-loop and closed-loop experiments.

This thesis begins with a discussion of the design of a testbed. The implementation

of the testbed, including the hardware used and the algorithms developed, are studied in detail. Chapter IV discusses issues encountered with camera calibration, both in hardware and software. The methods used are compared to other modern approaches to camera calibration [8]. The performance of the testbed is then analyzed for speed, accuracy and robustness to noise.

In the latter half of the thesis, Chapter VII discusses applications of this testbed, including the novel robust identification of a single mobile robot. Various controllers are designed and tested in a robot simulator and the recently constructed testbed. A controller is developed to provide a smooth trajectory from any point in the testbed to any other arbitrary point. A second controller is presented that tracks a circular path. The smooth path controller is later extended to coordinated movement between multiple robots. The methods shown here are tested in both a simulated environment as well as the testbed.

Following the applications, an extension to the current testbed is proposed in Chapter VIII to allow three dimensional tracking for aerial vehicles. Until recently, three dimensional testbeds have been built [5] strictly for outdoor use with large aerial vehicles. They employ expensive GPS and inertial sensors to measure the position of the robots. With the availability of small, affordable, aerial vehicles, three dimensional testbeds are being built for indoor use. MIT recently purchased just such a testbed from a company called Vicon [6]. Their new testbed uses high-speed, infrared (IR) cameras to track the position of multiple aerial vehicles. An alternate, low-cost design for a 3D testbed is presented here, which focuses on using small aerial vehicles in an indoor setting. It will also use a vision system to track the pose of multiple robots efficiently.

CHAPTER II
DESIGN OF 2D TESTBED

A. Design Goals

Before delving into the detailed design of the testbed, it is important to consider the objectives of this project.

1. Utilize the Lab Space Efficiently
2. Support Multiple Robots
3. Make Accurate Pose Measurements
4. Track Robots in Real-Time
5. Use a Generic Hardware Platform to Track Robots
6. Communicate Position Data to Mobile Robots

There are several requirements of this testbed. First, it needs to utilize the space in such a way as to facilitate experiments with multiple mobile robots. Therefore, the testbed could not be long and narrow. A square shape would allow several robots to move in different formations. It would also permit movement in any direction equally. A narrow testbed would constrain the majority of the movement to a single axis.

Second, the testbed needs to support multiple robots. It should be able to measure the position, the x and y coordinate, along with the orientation of each robot on the floor plane. The combination of the x and y coordinates with the orientation, θ, is referred to as the pose, written as (x, y, θ).

This testbed is made to emulate a local GPS system. GPS does not function reliably indoors. A GPS receiver detects time signals from four satellites. By measuring the time difference between each signal received, the device can calculate its position. Typical accuracies of GPS systems are reported in the range of meters. This testbed would be required to produce accurate measurements of each robot's pose in the centimeter range. Typical accuracies of other testbeds have been reported at ± 8 centimeters of the actual position and within 10 or 12 degrees of the actual orientation.

One critical objective of this testbed is to measure the pose information in "*real-time*". Real-time is a subjective term, but it is typically defined in process control systems as 10 to 60 Hz. The goal of this testbed is to perform at or near 30 pose measurements per second, which is the upper limit of this testbed because of the sensors used: NTSC cameras. The NTSC video standard is 30 frames per second.

Other testbeds have reported tracking frequencies of 30 Hz, but all of them perform the tracking using costly, specialized hardware. Another goal of this testbed is to use a generic software, sensor and computer platform. While saving money, implementing the tracking in software allows the testbed to be modified more easily to meet new requirements. Should a special situation arise, a new algorithm can be implemented to satisfy the new objective, all with the existing platform.

In order to make use of the testbed's pose measurements, there must be some method to communicate to the robots. The final goal of the testbed is to effectively communicate pose information to each robot. This testbed will provide a wireless platform to meet this need, but this communication method can be replaced or augmented at a later time.

B. Physical Testbed Design

This is a vision-based testbed, which means that cameras are used to perform the pose measurements. Video cameras are mounted in the ceiling and face downward toward

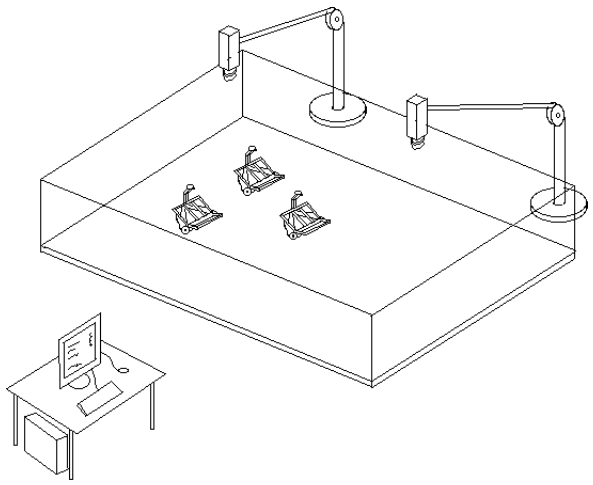

FIGURE 1 – A common setup for a mobile robotics testbed, using overhead cameras to track movements of robots on the floor below.

the floor. Four cameras are used to cover the entire testbed floor. In order to provide a high contrast background for the overhead cameras, the testbed floor is white. The shape of the testbed is a near-square shape to facilitate movement in all directions. It extends as large as possible while still accommodating other needs for the lab space and maximizing the coverage of the cameras. Fig. 1 shows an initial design of the testbed. Also shown in the design sketch is a desktop computer. The computer performs all of the pose measurement calculations from the images captured from the overhead cameras.

C. Software Algorithms

The needs of the testbed require the ability to track multiple robots in real-time. However, there are times when only one robot will be studied at time. As a display of the flexibility of the generic platform, two algorithms have been developed to meet two separate needs. The single robot algorithm takes advantage of the simplicity of searching for a single robot, increasing accuracy and the frequency of measurement. This can be used when only one robot is studied, yielding more accurate tracking. The multi-robot algorithm is inherently more complex. The search in this case must not only calculate the position

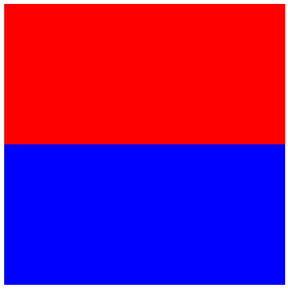

FIGURE 2 – Specialized color "hat" pattern worn on top of a single robot, to be identified by the overhead camera system for tracking. The red block is worn on the front of the robot with the blue on the back.

and orientation multiple times, but must also uniquely identify robots. The result is a more involved and therefore, slower measurement.

1. Single Robot

The algorithm to search for a single robot is simple. A specialized color pattern is placed on top of the robot, referred to as a "*hat*". With the robot's body hidden below the hat, the camera is searching for the appropriate pattern. The hat design for the single robot is shown in Fig 2. It consists of a red rectangle abutting a blue rectangle on the long edge. The hat is worn with the red rectangle on the front of the robot, while the blue is on the back. The algorithm is based on two separate center-of-mass measurements of the two colored rectangles on the hat. The center of mass of a two-dimensional object of uniform density - as in an image - is defined in equations (1) and (2):

$$x_{center} = \frac{\int\int x\, dx\, dy}{\int\int dx\, dy} \tag{1}$$

$$y_{center} = \frac{\int\int y\, dx\, dy}{\int\int dx\, dy} \tag{2}$$

With digital cameras, the mass distributions are discrete. So, equations (3) and (4) are more appropriate.

$$x_{center} = \frac{\sum\sum x_k\, m_k}{\sum\sum m_k} \tag{3}$$

$$y_{center} = \frac{\sum\sum y_k\, m_k}{\sum\sum m_k} \tag{4}$$

In a scan of the camera frames, the image is thresholded for red and blue. The x and y coordinates of each red pixel are summed. The x and y sums are then divided by the mass of the red pixels. This yields the two dimensional average of the red block, or its center. The same process is performed for all blue pixels. Once the centers of the red and blue block is found, it is easy to see that the average of these two centers will yield the center of the robot itself.

$$(x, y)_{Robot} = \left(\frac{x_{red} + x_{blue}}{2}, \frac{y_{red} + y_{blue}}{2} \right) \tag{5}$$

Using simple geometry, it can be shown that the arctangent of the line drawn from the red center to the blue center will give the orientation angle of the robot as shown in Eq. 6.

$$\theta_{Robot} = \arctan \left(\frac{y_{red} - y_{blue}}{x_{red} - x_{blue}} \right) \tag{6}$$

These measurements are taken in the global reference frame of the testbed. The origin of the testbed was chosen as an arbitrary corner. Figure 3 shows this global reference frame, along with the robot's local reference frame. In the global reference frame, the pose is typically represented by (X_G, Y_G, θ). The robot's local reference frame, (X_R, Y_R, θ), changes with the movement of the robot. A differential drive robot is shown in the figure. The origin in the local reference frame is mid-way between the wheels. The forward direction for the robot is in the X_R direction. Note that θ is defined the same in the robot's local reference frame as it is in the global reference frame.

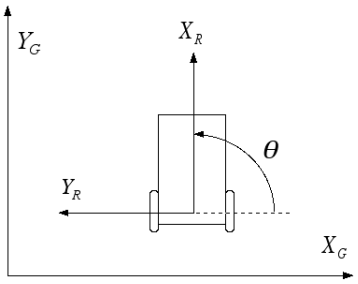

FIGURE 3 – Illustration of the reference frames for the robot. The testbed measures in the global reference frame (X_G, Y_G, θ) as defined in the figure. The robot's local reference frame (X_R, Y_R) is shown superimposed on the robot.

This algorithm is summarized in the pseudo-code below.

```
While tracking the robot {
    For each pixel in image {
        If the pixel is Blue
            Add the X coordinate to the blue X summation
            Add the Y coordinate to the blue Y summation
            Increment the mass of the blue pixels
        If the pixel is Red
            Add the X coordinate to the red X summation
            Add the Y coordinate to the red Y summation
            Increment the mass of the red pixels
    }
    Divide the X and Y blue summations by the blue mass
    Divide the X and Y red summations by the red mass
    Average the results to find the center of the robot
    Calculate θ
}
```

This algorithm has several advantages. First, it is very simple, which provides a faster run-time. It only involves two summations, a couple of division operations and an arc-tangent calculation. It also only requires a single pass over the image. Most computer vision algorithms require multiple loops over every pixel. Even with fast processors, scanning an image with hundreds of thousands of pixels is a time consuming process. A great effort is made to consolidate operation loops. In this case, there is only a single pass over the image, increasing the measurement frequency.

2. Multiple Robots

The algorithm to track multiple robots is more complicated for several reasons. Now, the process of measuring the pose must be repeated for each robot that is found on the surface. Also, once the locations and orientations are found, they must be matched to the correct robot. So, a third operation of uniquely identifying each robot is added to the process.

To accurately measure and identify each robot, a new pattern system is needed. The simple hat could be extended for the use of multiple robots, but only to a point. With the simple algorithm, each robot would require a hat with two unique colors. One would replace the red block while another color would replace the blue. In order to differentiate between the robots, the colors themselves would need to be distinguishable to the cameras overhead. If the testbed needed to measure eight robots, sixteen different colors would be needed for the hat patterns.

In theory, this is possible. However, in practice, there is often too much noise in an image to classify closely related colors accurately. As a robot moves around the testbed floor, there are slight variations in lighting. Light condition changes have long plagued computer vision research, as many algorithms are susceptible to them. Although there are many statistical, and even spatial, techniques for accurate classification, these methods are far too slow to meet the real-time demand of this system.

This method of using unique colors to identify each robot also reduces the scalability of the testbed. There are a limited number of colors that can be classified uniquely. Depending on the noise level in the system, the number can be as high as 32. This would allow for only 16 robots. For a large testbed, this number might not be sufficient. For most practical systems, the number of unique colors is closer to eight or nine, only allowing four or five robots on the testbed.

Instead of simply extending the single robot algorithm, an entirely new one was developed. The design template for the hat pattern is shown in Fig. 4. It is only a template,

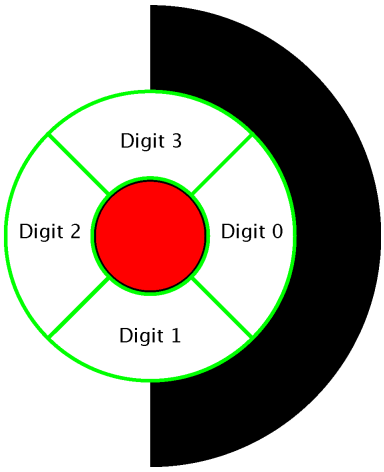

FIGURE 4 – Specialized color "hat" template pattern for multiple robots. The red circle in the center is used for the x & y calculation. The black/white semi-circles on the outside are for the orientation measurement. Four quarter circles surround the red center, shown here in green, are each filled with one of three colors (Green, Blue and White) to encode the robot number into the pattern making each hat unique.

because each hat will have a unique identifier pattern associated with each robot. There are some common features among each hat pattern. The first is the central red circle. Red is a color that has proven to threshold easily and is used here to initially detect a robot.

To save scanning time, the image is sub-sampled to detect a red pixel. In other words, only one pixel will be examined within a neighborhood of several pixels. That pixel is thresholded. If the pixel is some color besides red, it is set aside and a pixel from the next neighborhood is studied. This procedure continues skipping across a grid over the image until a red pixel is located. When this happens, the pixel neighborhood is examined for other red pixels. The center of mass is then calculated for the red pixels discovered in the neighborhood. This gives the center of a robot. At this time it is still not known which robot it might be. Once the center of the red circle is found, the second feature of the hat pattern is used to determine the orientation of the yet-to-be-named robot. The black and white semi-circles surrounding the pattern provide unique points that can be used to find the orientation. Those unique areas are the beginning and end of the semi-circles, or where the black and white shapes meet each other. By moving around the robot's center at a constant radius, the pixels are thresholded until the black-white or white-black edge

is found. This will give a second point of reference for this robot. This point, along with the center, can be used to calculate the orientation angle, θ, in the same way as the simple algorithm. It is simply the arctangent of the line constructed from these two points.

Now that the location and orientation have been found for the robot, it is time to identify which robot it is. The third feature of this pattern is the inner ring area. Here, the inner ring, outlined in green in Fig. 4, is divided into four equal regions or quarter circles. The robot's number is encoded into this ring using three colors: green, blue and white. Green represents the number two. Blue represents one and white makes zero. With three colors, the encoding scheme is base 3. The four sections of this ring are filled with the different colors to make up four digits in the scheme. So, the number of unique hat patterns is 3^4 which is 81. This means the software could support up to 81 different robots. It is safe to say that this number will not be reached, but still leaves plenty of room for expansion.

Once a robot's pose variables have been found, along with its identifier, the search continues across the rest of the image for red pixels. One note here is that care must be taken not to search through previously discovered red pixels. This would lead to calculating the pose variables of an already-discovered robot. To prevent this, once a red pixel is found, its value is changed to an impossible level. This guarantees that it will not threshold as red again.

D. Wireless Communication

The initial design of the wireless communication scheme uses a standard protocol known as Universal Datagram Packet (UDP). The options were UDP and Transport Control Protocol (TCP). The UDP standard was selected for its simplicity and flexibility. UDP sends messages as a broadcast. TCP is more complex in that it requires a relationship to be set up using "*hand-shaking*" with the intended receiver of the message. The protocol sends a message to a receiver and then waits to hear back from the receiver, signifying the message was transmitted correctly. With UDP, there is no hand-shaking, or any request of

information back from the intended receiver. The message is sent without worrying about the receipt of the message. This involves less overhead for both the vision system and the robots themselves.

For example, a robot may be busy processing data from a sensor mounted on top of it. It may be perfectly still while it makes a decision what to do next. If the robot knows its pose, it would be a waste of time to "listen" for an update. When the robot needs an update of its pose, it can listen for the next broadcast.

Using UDP, the vision system broadcasts the pose information without worrying if the message was received. Consequently, this scheme is less reliable than TCP communication, but speed and simplicity are gained using UDP; both for the vision system and the robot.

CHAPTER III
IMPLEMENTATION OF 2D TESTBED

This chapter covers the implementation details of the testbed. It begins with the hardware used and closes by discussing software issues.

A. Hardware Implementation

1. Testbed Floor Construction

The lab space given needed to house student researchers as well as the testbed. In order to utilize as much space as possible, the testbed floor stretched wall to wall. Two different configurations are shown in Fig. 5. The first was chosen because it provided more of a square shape while using the space most effectively.

An odd material was used for this floor: dry-erase marker board. The original flooring of the lab was a multi-colored tile designed to be anti-static. This is important for work with small scale electronic devices, such as clean rooms. However, this application depended more on the color of the floor. A single, high-contrast color would allow the overhead camera system to more quickly distinguish between a potential mobile robot and the background. A multi-colored floor would require more image processing. Although the processing would be very simple, it would waste precious time and considerably decrease the frequency of the testbed's measurements. This would jeopardize the real-time goal of the testbed.

The only other requirement of the flooring used was that it would provide enough friction to prevent significant wheel slippage. A dry erase board was tested before construc-

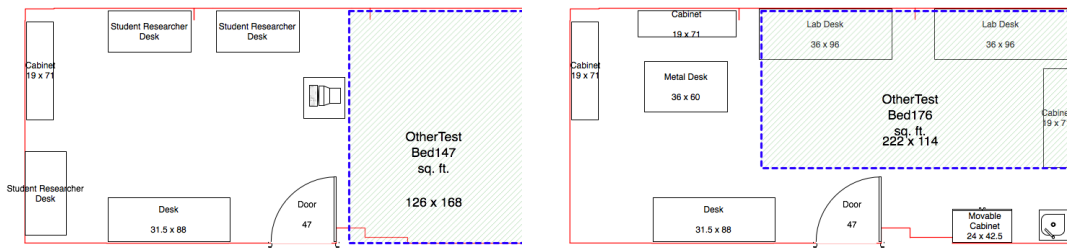

FIGURE 5 – CAD drawings of two possible configurations for the laboratory space. The first layout was chosen because it more efficiently utilized the space.

tion began. Whether the board was dusty or clean, it still provided a considerable amount of friction for the robot's wheels. The same was true when the test was repeated for dirty robot wheels.

In order to protect the underlying anti-static tile, a sub-floor was anchored into the concrete. Eighth-inch plywood was used to construct the sub-floor, spanning the entire area of the testbed. The seams of the sub-floor were taped in an attempt to smooth the edges of each seam. The sub-floor panels were scribed to meet all of contours of the outer walls.

The white marker board sheets lay perpendicular to the sub-floor panels so that the seams of the sub-floor would not meet the seams of the marker board sheets. This would help minimize ridges where the panels met. The top floor was glued to the sub-floor to avoid using nails and potentially cracking the top surface. The edges and seams of the top floor were filled with a white caulk to prevent moisture and dust from getting under the surface. It was later found that the caulking attracted dust particles, turning the caulk a dark brown color. In order to facilitate cleaning the testbed, the caulk lines were painted flat white. This successfully prevented dust from collecting at the seams.

The rubber base around the wall was also found to collect dust and dirt. This caused the occasional pixel to threshold to red or even green, producing inaccuracies in the robot

FIGURE 6 – Image of actual testbed floor. The short fence is in the bottom of the image. The walls enclose the other sides.

measurements. To solve this problem, the trim was also painted flat white.

Three sides of the testbed are enclosed by walls. The fourth side is closed with a short fence to prevent the robots from accidentally driving outside the testbed. Figure 6 shows the finished testbed floor.

2. Camera Location

Four video cameras are mounted in the ceiling, facing downward. They are offset in a way that maximizes the floor coverage. A model of the testbed was built in Octave to simulate the placement of the cameras as well as their floor coverage. One rendering of the model is shown in Fig. 7. The height of the cameras is the main variable in this modeling problem. Once the height is determined, the placement positions of all the cameras are calculated so that the entire testbed area is covered. There is a light that spans the width of the testbed that hangs 12 inches below the ceiling. It was a potential obstruction for the cameras, so it was included in the model. From Fig. 7, the floor of the testbed is shown as four overlapping rectangles. The corners of these rectangles correspond to red lines

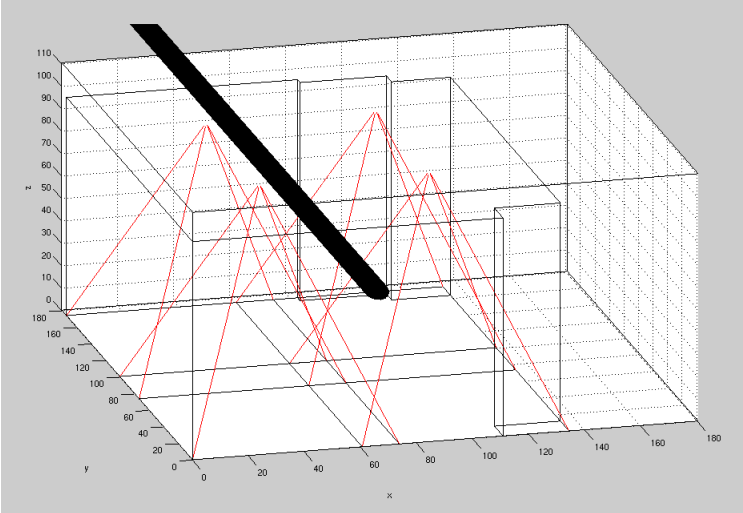

FIGURE 7 – Three dimensional model used to simulate placement of the cameras in the room. The large, black bar in the model represents the lights that hangs 12" below the ceiling. Measurements calculated from this model were used to construct the physical testbed.

extending from the cameras showing the fields of view. The angles of these red lines were found experimentally and will be discussed later with camera calibration issues. The height of the cameras could be varied, and the rectangles, or fields of view, of each camera could be analyzed. The model would also show if the hanging light would obstruct any camera view. Once an appropriate height was found, the position parameters from the model were used to place the physical cameras in the ceiling.

In the initial testbed design, the frames from each of the four cameras would be cropped and joined together to form a single large picture of the testbed. This allowed for a simpler search algorithm. In order for this to take place, the camera coverage needed to overlap, for several reasons.

First, when a robot passes between cameras, it is more effective if the cameras have a common reference. It is virtually impossible to align the cameras perfectly with respect to each other. In any practical case, there will always be some offset which creates a "*schism effect*". This is best illustrated in Fig. 8. Here, two robots are shown in one section of the testbed. The robot on the left is moving between two cameras and its hat pattern is altered. This could cause the algorithm to fail. The robot on the right is traveling through the center

FIGURE 8 – A gray-scale image of a section of the testbed with two robots. The cameras were carefully tuned so that they would combine well in the center of the testbed. The robot on the right is in the center and there is a nice image of the pattern. Although the image is nice here, that is not the case for another area of the testbed. The robot on the left shows the *schism effect* of lens distortion.

of the testbed. The four camera frames join here and actually produce a nice image of the center robot. Although the center is aligned well, the left side of the image is not. This introduces a more serious problem: *lens distortion*. Lens distortion is present in all lenses. This particular type of lens distortion is known as *barrel distortion*. Figure 9 shows an example of this distortion. Barrel distortion is a phenomena observed with curved lenses, where the magnification decreases as the distance from the center of the lens increases. In the figure, the grid lines should be perpendicular, but the image was taken through a common, curved lens, causing the lines to bow. The grid squares in the center of the image are slightly larger than those on the outer edges; especially the corners. Barrel distortion is more prevalent in wide-angle lenses, but even the video cameras used here, though not wide-angle lenses, produce some distortion. The distortion is most apparent in the corners of the video frame.

Unfortunately, the corners of the frames need to match when all four camera image frames are combined to form a single image. This problem can be lessened with larger overlap. It allows the higher distortion corner regions to be cropped. Areas closer to the center of the lens have far less distortion, so images can be connected much more smoothly. Figure 8 proves that there will always be a rift present when trying to join images together.

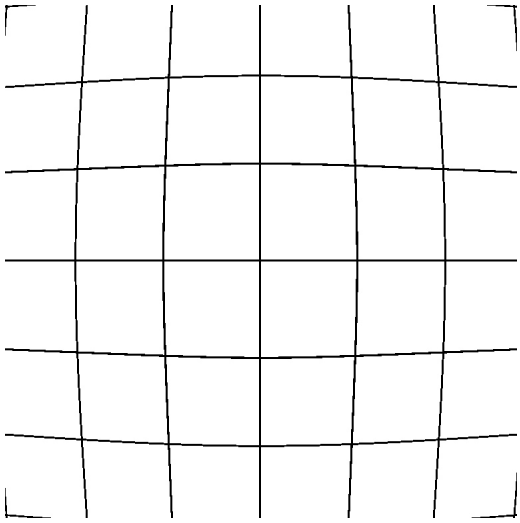

FIGURE 9 – This is an image taken through a lens with *barrel distortion*. The grid lines should all be orthogonal. There is a difference in magnification levels between the center of the lens and the outside of the lens.

If the system is tuned for one area to match, it will cause a greater schism in another area. All of the distortion effects could be mitigated more easily if the cameras overlapped enough so that the entire robot could be seen by both neighboring cameras when traveling between regions. This would eliminate the need to join the images at all. Neighboring cameras could get a full view of the robot in the transition region. A robot's position could be calculated independently by two cameras while in the overlap region.

The model clearly showed this would not be possible unless the cameras were recessed into the ceiling. It was desirable to keep the cameras below the ceiling. So, initially they were mounted that way. Later, the lens distortion proved to be too great for the details of the multi-robot pattern. When the robot traveled between camera regions, the algorithm would fail unpredictably. The best solution was to raise the cameras to a height that would allow a robot to be completely seen by at least two cameras when traveling between regions of coverage. The model is shown in Fig. 10. Notice how the model shows the hanging light is just outside of the field of view of the cameras. This provided a maximum allowable height for the camera mounts. The ceiling height is 110" from the floor. The camera lenses are mounted 112" above the floor.

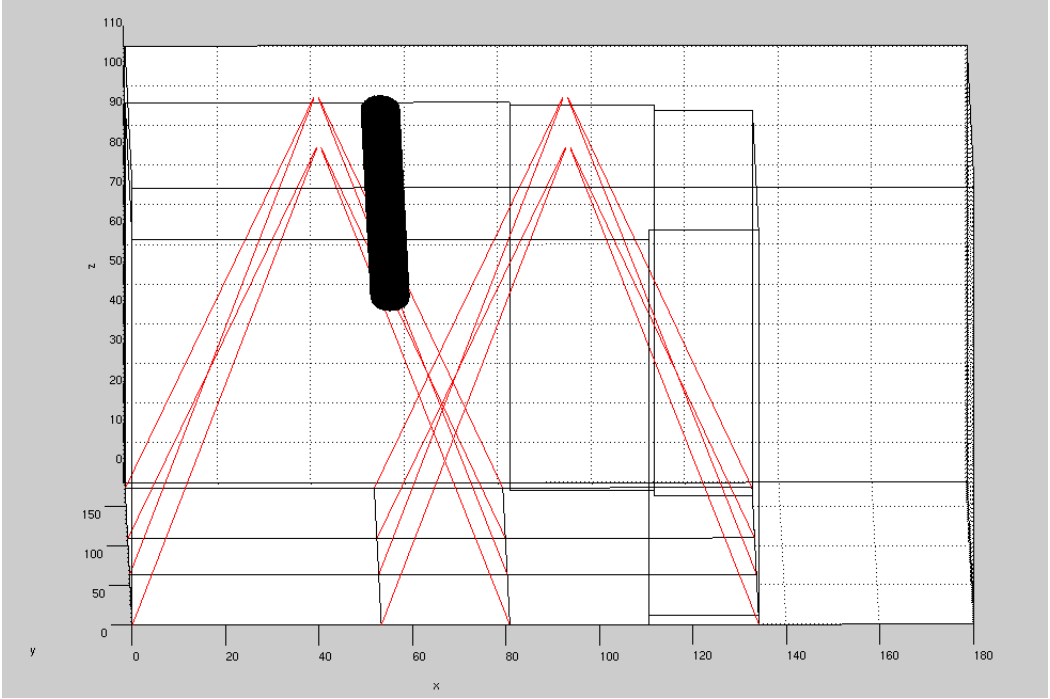

FIGURE 10 – Testbed model shown at camera height used for actual testbed. With respect to the origin shown in the model, the camera lenses are located at the following positions: (36.8”, 50.0”, 112.0”), (36.8”, 123.5”, 112.0”), (97.2”, 50.0”, 112.0”), (97.2”, 123.5”, 112.0”)

3. Camera Mount Construction

Although the computer model showed a fairly accurate rendering of the testbed, there will always be some disparity between the model and the actual placement of the cameras. This was taken into account when building the camera mounts. Another aspect considered was the large amount of obstructions above the ceiling. The mounting system would need to be flexible to avoid obstructions and adjust camera positions to cover the testbed floor.

The structure consists of two beams running parallel the length of the testbed. Smaller platforms span the separation of the beams. The platforms and the beams are joined with threaded rod. The long beams have a slotted hole that runs the length of the beam, allowing the threaded rod to be joined at any place on the beam. The platforms actually support the cameras. A similar slotted hole runs the length of the platform, allowing

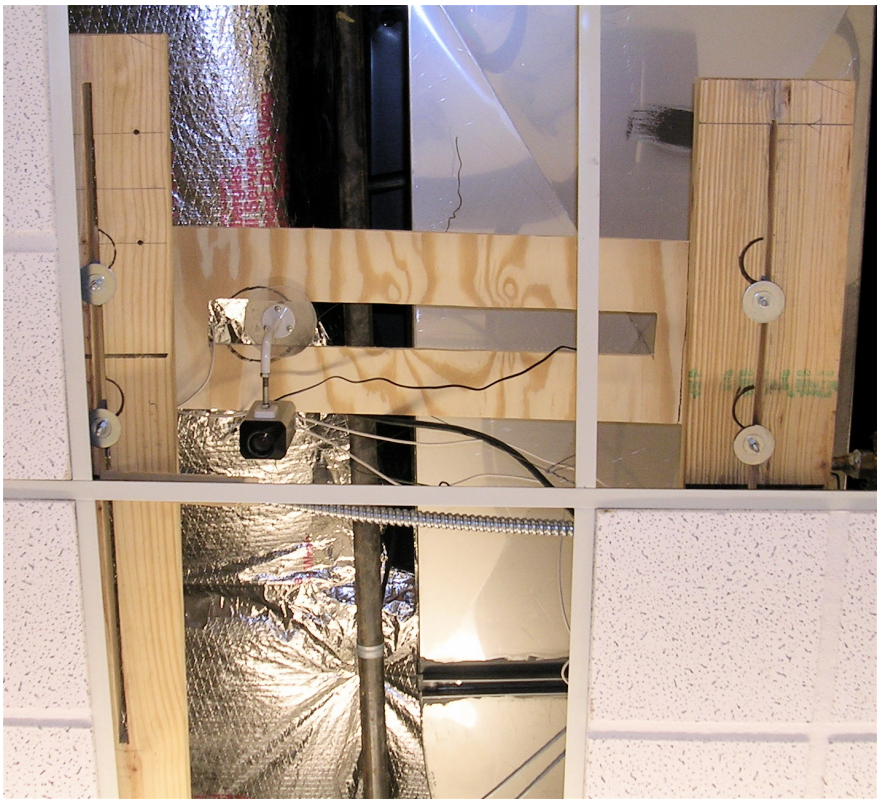

FIGURE 11 – Mounting beams used to support cameras. Slotted holes allow them to be mounted anywhere along the beam for flexibility during calibration.

the camera mount to attach at any point along the platform. These two slotted sections allow the camera to pan in the x or y direction. These slotted-hole sections are shown in Figs. 11 & 12. The threaded rod connections provide vertical adjustment for the cameras. By simply loosening or tightening a set of four nuts, the camera platforms can be raised and lowered, as well as leveled. It is important to have all of the cameras level so that perspective distortion is negated. An angled camera would produce an effect known as "*converging verticals*", where parallel lines appear slightly angled in the image. This is most apparent with much larger distances, however, care was still taken to avoid it.

Consideration was also given to rotation. The camera L-joints are joined to the platforms by two discs that "sandwich" the platform. The discs are bolted together in the center, which can be seen in Fig. 13. When loosened, the discs can slide any way along the slotted hole, providing the panning motion described previously. The discs can

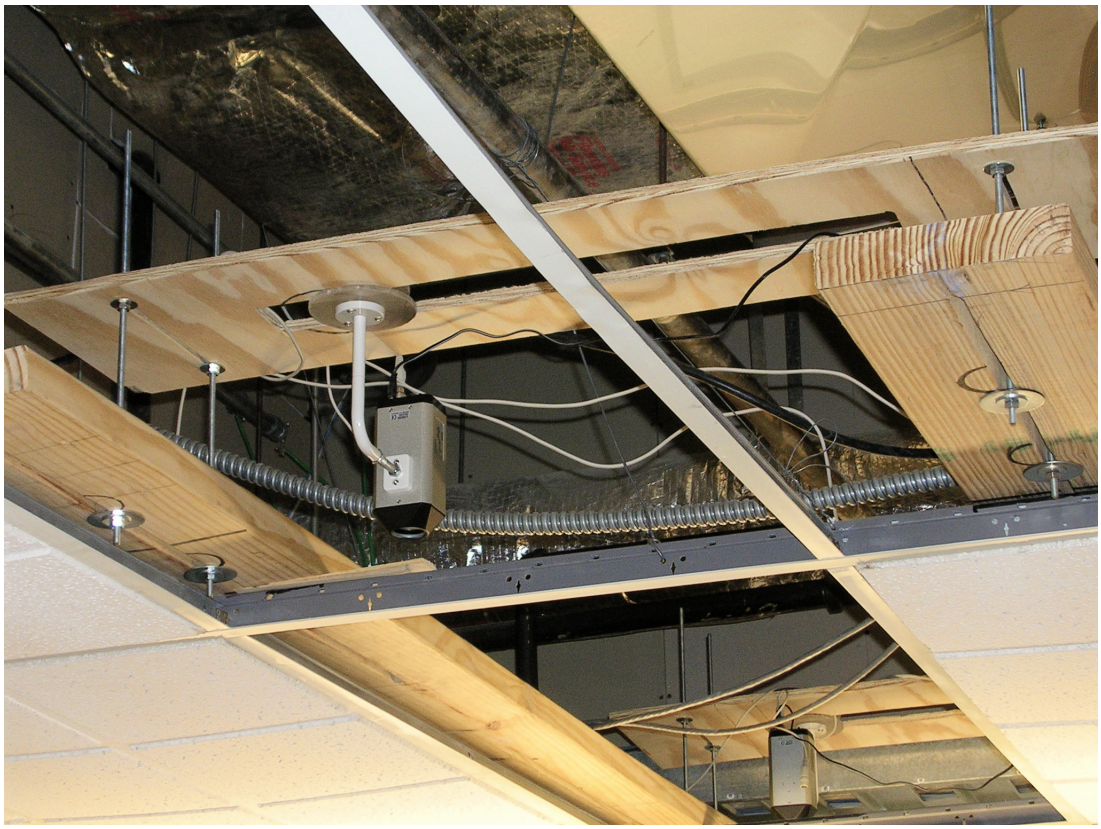

FIGURE 12 – Mounting platforms that hold cameras. Slotted regions allow the cameras to slide anywhere along the platform.

also be rotated, allowing the camera to be rotated squarely with the floor. Two cameras share two long beams, with their respective platforms spanning them at opposite ends. Figure 14 shows the final assembly with the cameras in place. With this configuration, the cameras can see the entire robot as it passes between camera regions, so there is no need for combining distorted images. This figure shows two pictures of all the testbed camera mounting hardware.

B. Software Implementation

The language chosen to implement the tracking algorithms is C++. This was chosen for its compatibility with known drivers, as well as robotics software. The source code for the single and multi robot algorithms is located in Appendix I and II, respectively. The

FIGURE 13 – Close-up of mounting brackets or discs used to attach L-arm to mounting platform. The discs sandwich the platform, providing a solid base while still allowing the cameras to rotate, if needed.

algorithms presented previously do not differ with those actually implemented. However, there are a few implementation details worth mentioning. The problems encountered along the way, along with the solutions, are also discussed.

1. Efficient Enhancements

Two common features of both the single and multi-robot algorithms greatly improved the speed of the robot searches. One feature was the thresholding method. Typically, there is a sequence of "If Greater Than Low Level And Less Than High Level" statements that determine whether a pixel gets classified as a particular color. For color images, there are three of these If-type statements for every color. This is not very efficient, especially as the number of colors increase. This can quickly become a time consuming process.

Another method involves using lookup tables for threshold values. Three vectors are used to represent three color channels. The ranges corresponding to an acceptable level are assigned a "1", while the ranges not associated with the color are "0". Figure 15 shows

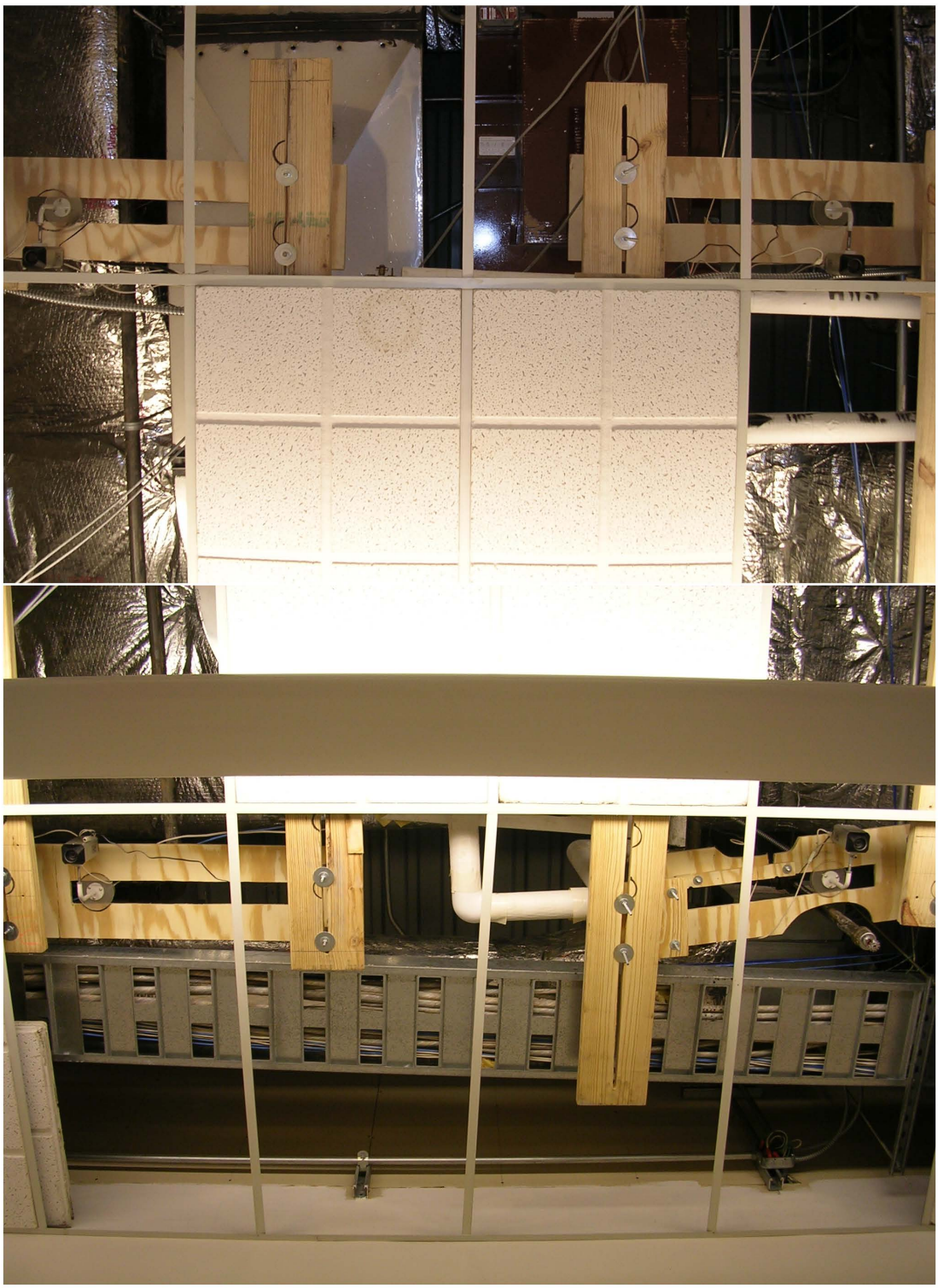

FIGURE 14 – View of all the camera mounting hardware for the testbed.

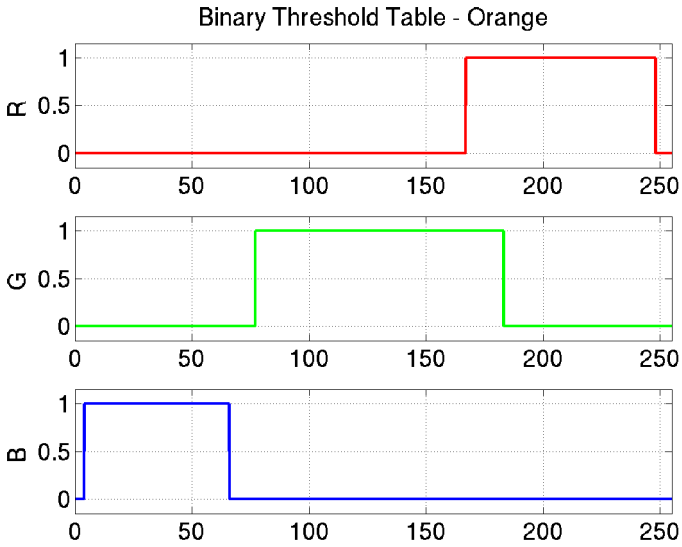

FIGURE 15 – Example of a thresholding lookup table for the color orange in the RGB color space. 1's represent an acceptable match level. If the lookup table returns three 1's, the pixel will be classified as orange.

an example threshold table for the color orange. In the Red-Green-Blue (RGB) color space, orange is represented by a mixture of red and some green. Color values are typically 8 bits, which yield 2^8 or 255 different levels. So, the threshold tables have a length of 255. Higher values of the red channel are assigned "1", along with the mid-level green values. Orange has very little blue content, so "1's" are given to the lower blue levels. The thresholding is performed using the pixel vector as the indices to the thresholding tables. The result of the three look-ups will be a mixture of zeros and ones. These results are ANDed together to give the final result. If the pixel vector finds a "1" in all three lookup tables, the pixel is a match to the color orange. If one of the pixel components finds a "0", the pixel will not be classified as orange. This algorithm has been found to drastically reduce computation time for thresholding an image [9].

The typical compiler assigns 32 bits for an integer. The lookup tables only require a single bit in each of them to represent a color. With integer lookup tables, 32 color threshold levels can be stored. One thing to note with this method is that overlap of colors in the lookup table is possible. One pixel could be classified as being red or orange if there is

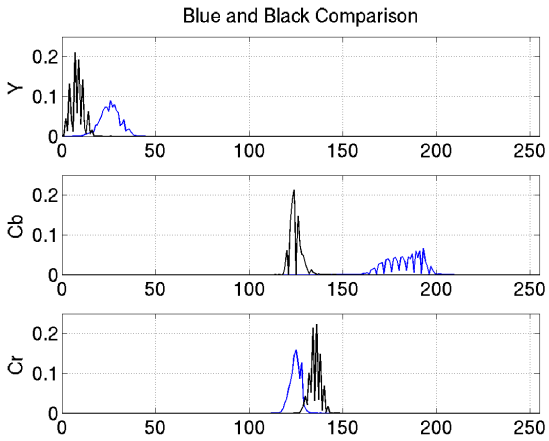

FIGURE 16 – Histograms of the colors blue and black in the YCbCr colorspace. Notice the overlap in the Y and Cr channels. There is also little separation in the Cb channel, making these two colors difficult to distinguish, especially in changing lighting conditions.

overlap in the lookup table. This is an acceptable situation, because the color patterns were carefully selected in order to maximize the contrast between colors in the same feature.

For example, blue and black are very closely related according to threshold levels. Figure 16 shows histograms of the YCbCr components for the two colors. The YCbCr or YUV colorspace is another way to represent color and will be discussed in more detail later. Notice that there is some overlap, particularly in the Y and Cr component. Blue has a slightly higher, though not always distinct, gray-level. It also usually has a higher blue component. When there is noise present in the environment, blue and black become difficult to separate. However, blue and black are never used in the same feature. Black and white are used for the orientation measurement. Blue, green and white are used to encode the robot number. So, classification error due to the overlap of black and blue threshold regions will not cause the algorithm to fail.

The second beneficial commonality of the single and multi-robot algorithms is the use of sub-sampling during the image search. This method is used successfully in [7]. Instead of analyzing every single pixel in the image, varying numbers of rows and columns are skipped. Whenever a critical color is found, such as red or blue, a more detailed search

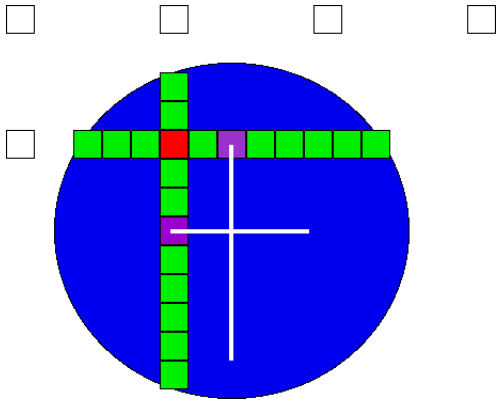

FIGURE 17 – Search method and center calculation used in [7]. The sub-sampled search method was used for the local testbed, but a different algorithm was chosen for pose calculations. White squares indicate not finding the blue circle. Red square indicates a "hit" from the search for blue. Green pixels are measuring chords of the circle. Purple squares are the center of each chord. White lines show perpendicular bisectors, which intersect at the center of the blue circle, a.k.a. the center of the robot.

is performed for that area of the image. In [7], they use a pattern very similar to the pattern used in this multi-robot algorithm. All hat patterns developed have the same features: 1) An area to locate the center of the robot 2) Some sections to encode a robot ID number 3) A feature to find the orientation. Although the patterns are similar, the algorithms are quite unique. The method used in [7] uses a sub-sampled search, but a entirely different method is used for finding the robot's center. The centroid calculation used here is illustrated in Fig. 17.

The white squares indicate the sub-sampled search from left to right, top to bottom. The pattern used here searches for a blue central circle, which is shown in the figure. The red square indicates a "hit" from the sub-sampled search. Now, the search expands to the north, south, east and west, as shown by the green pixels. This search stops when the edges of the blue circle are found. The green lines are perpendicular chords of the blue circle. Perpendicular bisectors of the chords, shown as white lines in the figure, will intersect at the center of the blue circle. This gives the center of the robot.

Although this method requires a search of very few pixels, it was not the method chosen to track the robots for this testbed. This approach is very fast, but is highly susceptible to noise in the environment. If a single pixel in the north-south or east-west line is

misclassified, the centroid calculation will be inaccurate.

Instead, once a "hit" is found during the sub-sampled searching, a thorough neighborhood search is performed and the center of mass is calculated as described by equation 5. If a single pixel is misclassified, the centroid calculation does not suffer. Figure 18 shows a visual representation of this method for the single robot hat pattern. The white pixels in the figure are not analyzed for color content. They are assumed to belong to the background. The black dots show which pixels actually were measured for color content. They were set to black when they did not threshold to red or blue. The search continues across the image until a red or blue pixel is found. There is a large section of black pixels around the robot in the image. This is a result of finding one of the red pixels in the image. The black box represents the neighborhood of pixels searched after that event occurred. Some of the pixels in the black box thresholded to blue and some to red. The centers of mass were calculated for these two blocks. Green dots were placed at those centers of mass. The final position is the result of averaging the red and blue centers, as discussed previously.

The searching box can be varied in size, but must be large enough to encapsulate the entire robot. For the single robot algorithm, the search box is large because it must encompass the entire hat. The box is smaller for the multi-robot hat because all that needs to be found is the small red circle. Also, the number of rows and columns that are skipped changes between the algorithms. It must be kept small enough to be sure to detect at least one critical color pixel. For the single robot, the red and blue blocks are quite large, so more rows and columns can be skipped, saving time. However, the multi-robot pattern contains a smaller critical area, so fewer rows and columns are passed over in order to guarantee detection.

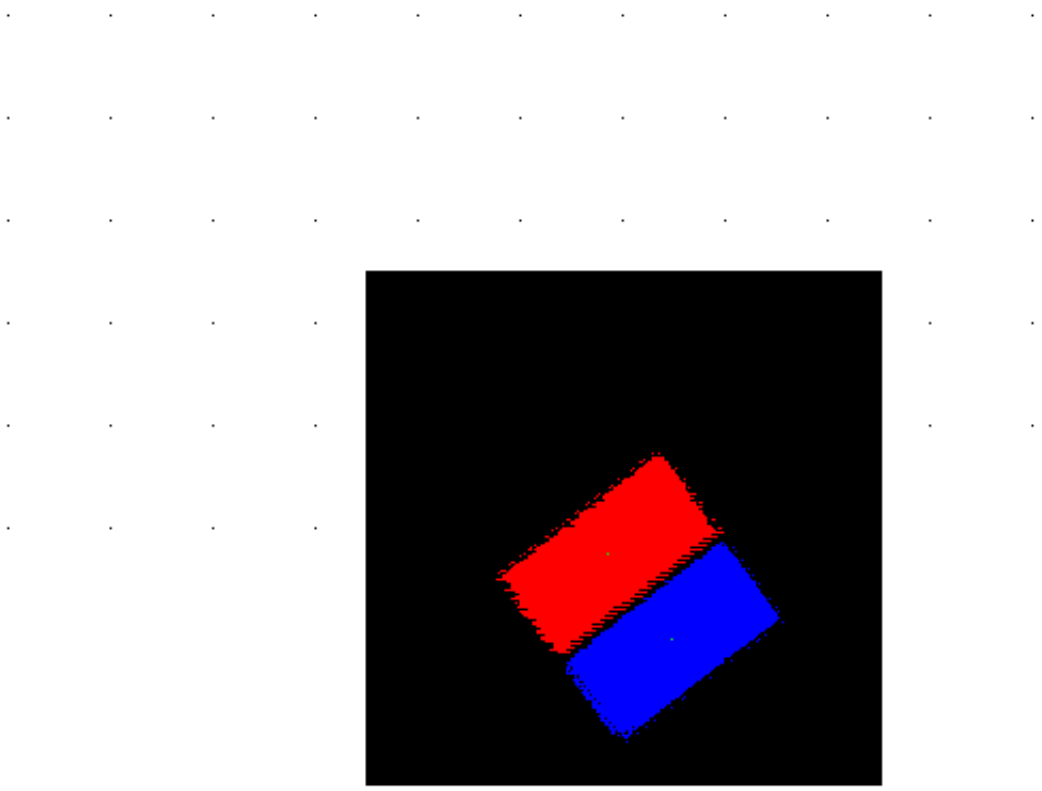

FIGURE 18 – Visual representation of the sub-sampled search algorithm. White represents pixels that were not examined. The black dots show what pixels were thresholded. The black box shows where the refined area search took place. This is the search for the single robot. The green dots represent the centers of the red block and the blue block. Notice that the algorithm stops searching once the single robot is found.

2. Challenges

Between the two algorithms, the single robot and multi-robot, several challenges arose. The first implementation of both of these algorithms fell far short of expectations. While both accurately measured the pose of the vehicles, the measurement frequency was a mere 1.5 Hz. This failed to meet the real-time need of the testbed.

After further analysis, it was found that both programs spent most of the time waiting for the image data to arrive to memory for processing. The robot search algorithms operated a scant 8% of the running time. The other 92% was spent waiting for the frame-grabber to capture an image. The frame-grabber purchased was capable of incredibly fast capture rates, however, the problem was the software driver.

After much investigation into the workings of the driver, it was discovered that the frame data went through a time-consuming color-space conversion. The native output format of the cameras is YUV, a color space consisting of gray-scale (Y), blue component (U) and red component (V) channels. The driver was converting the YUV channels to Red-Green-Blue (RGB). Then, it performed a second conversion from 4-bit RGB values to 8-bit RGB values. Once an image was captured, the pixels were converted to two different color-spaces. Only then could another image be captured.

At this point the search algorithm could begin, but too much time was already lost. The end result was a sluggish tracking rate of 1.5 Hz. So, a new driver was selected that would capture the frame data more efficiently. This new driver had the option to work in the native format of the camera, YUV. The driver was modified slightly to get even faster performance. The output directly from the camera was in a compressed video format known as YUV 4:2:2. This format is similar to that used by high-definition televisions. The frames were analyzed in this raw, compressed YUV 4:2:2 format to greatly improve the measurement frequency of the testbed. As a result modifying the frame-grabber drivers, the tracking rate surged from 1.5 to 30 Hz.

The second major challenge was mentioned earlier with the camera mounting. The

lens distortion of the cameras prevented the multi-robot algorithm from measuring robots in the overlap region. The single robot hat pattern is large and simple. Small schisms or rifts in combining four camera frames together did not affect the end result of the simple algorithm. The mere size of the simple hat features made it robust enough to handle well.

The multi-robot hat pattern contains more information. In order to keep the pattern roughly that same size as the single robot pattern, the feature size was reduced. This magnified the lens distortion's effects. Suddenly, the search algorithm would find a 70 degree shift in the robot's orientation simply because the images from neighboring cameras had some sort of rift.

This was not a problem that some amount of calibration could correct. Even if cameras were calibrated to provide perfect matching in the center of the testbed, the outer edge of the testbed would show the distortion effect. This would cause the multi algorithm to fail. The only solution was to raise the camera height. Using the model shown previously, the cameras were raised from a height of 92" above the floor to 112". The overlap area increased enough so that the need to combine frames was eliminated.

CHAPTER IV
CAMERA CALIBRATION

Several problems arose while working with the cameras. In practical applications, a camera, like anything else, is imperfect. Lens distortion, though small, proved a formidable challenge in the implementation of this testbed. That problem, discussed previously, was solved by raising the height of the cameras. Another problem was reconciling the differences in color levels between each camera.

Although the four cameras are identical in make and model, they still do not respond exactly the same way to light. The charge-coupled device (CCD) sensors employed on the cameras, though similar, are not exactly the same. There are several parameters on the cameras that can reduce these differences such as color gains and the aperture setting. However, it was still not possible to get identical color levels from the four different cameras.

Another difficulty encountered dealt with color thresholding. Color thresholding is a process of classifying a pixel as a particular color. There are very complex and time consuming methods of pattern classification [10]. Typically, the more advanced the method, the more costly it is in terms of speed. One of the major goals of this testbed was to operate in real time. So, a great effort was made to separate the colors as much as possible.

The camera hardware settings and color thresholding will be discussed in detail later in the chapter. The first feature studied is a simple one. The object of using overhead cameras was to track robots on the testbed floor. In order to know how much area a single camera could cover, the field of view of the camera was measured.

A. Field of View

The ultimate goal of using the cameras was to sufficiently cover the testbed floor and track vehicles, wherever they might be. So, the field of view of a single camera was calculated experimentally. This was done by taking measurements of the camera's cone angle. Light enters the lens at an angle and is refracted to a focal point. A rectangular array of light sensors is placed between the focal point and the lens, sensing the incoming light. Any light entering the lens greater than a specified angle is not refracted through this system and therefore unseen by the image sensor.

This angle is referred to as the *field of view* or *cone angle*. This angle was measured experimentally at specific distances from the camera lens along the horizontal plane of the camera, as well as the vertical plane. The camera was placed on the floor, centered on a straight line. At a distance of 10 inches from the lens, an object was placed on the floor, outside the camera's view. Using live video from the camera, the object was moved toward the center line. When the object was seen by the camera, the perpendicular distance from the center line was recorded. This process was repeated in 10 inch increments to 70 inches. This was also done for the opposite side of the center line in the horizontal plane. The angle found was symmetric about the center line. The total field of view in the horizontal plane measured 52.79°.

The camera was then rotated 90 degrees to take measurements in the vertical plane. The process was repeated. It was discovered that the vertical angle was different from the angle in the horizontal plane, at only 40.16°. This is expected because the CCD sensor is not square. There are 640 lines of resolution that span the horizontal axis, while only 480 cover the vertical axis. These measurements were plotted in three dimensions. Figure 19 shows this plot. The red circles represent the data points in the vertical plane. The blue represent the horizontal plane measurements. Using the least squares method, lines of best fit were found and plotted as green lines. The final camera coverage is shown as the green rectangle at a distance of 110 inches from the camera lens. These measurements were used

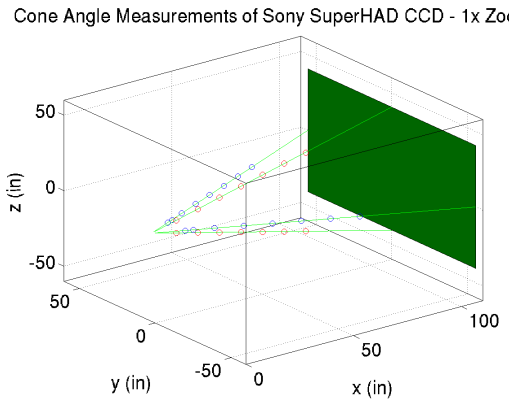

FIGURE 19 – 3D view of cone angle measurements taken from video camera at 1x zoom. Red points represent measurements taken in the vertical plane. Blue points are measured in the horizontal plane. Green lines show least squares fit. Rectangle shows field of view at a distance of 110".

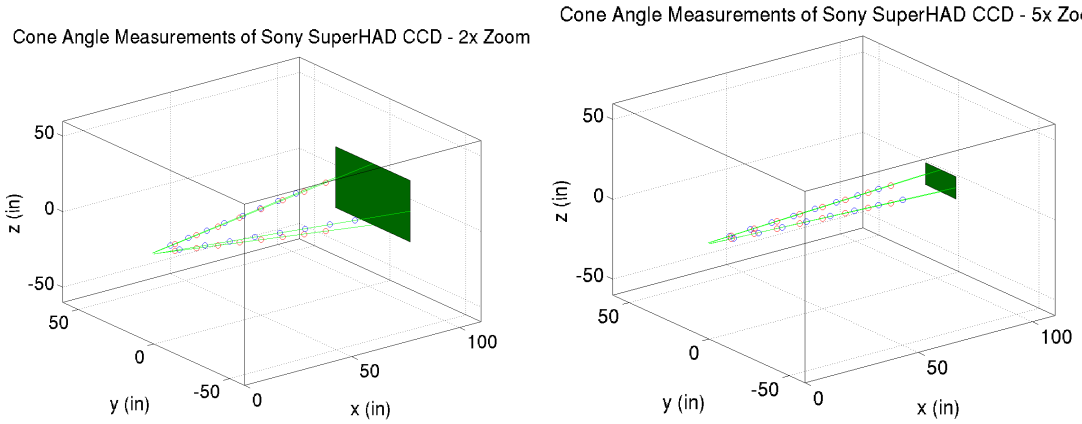

FIGURE 20 – 3D view of cone angle measurements taken from video camera at 2x Zoom on the left, 5x Zoom on the right. Red points represent measurements taken in the vertical plane. Blue points are measured in the horizontal plane. Green lines show least squares fit. Rectangle shows field of view at a distance of 110".

to develop the testbed model seen previously in Chapter III. The measurement process for 1x zoom was repeated for another camera and the results were indistinguishable at the level of measurement precision. So, the model here was applied to all cameras.

These measurements were also repeated for different zoom levels of the camera. The original experiment was at 1x zoom. The next two experiments were done at 2x and 5x zoom. The results are plotted in Fig. 20. Notice how the zoom affects the field of view. The angles are decreased as the zoom increases, allowing less area to be covered.

FIGURE 21 – Picture of Cardinal quarterback Brian Brohm throwing a pass against the Syracuse Orangemen. Used here to illustrate the YCbCr color space. Left, is the original image. The images to the right of the original are the Y, Cb and Cr represented as gray-scale images.

B. Multi-Camera Color Consistency

Although the cone angles of the four cameras matched well with each other, some of the other features did not. Of the twenty or so adjustable settings on the cameras, only three of them proved to have a major affect on the output: aperture, red gain and blue gain. The native output of the cameras is YUV or, once digitized, is also known as YCbCr. This is known as component video and is a widely used format in video processing [11].

1. A Word About Color Spaces

The Y component represents the gray-scale or intensity component of the image. It is simply the black and white image. The other two channels contain the color information of the image. They are represented as the blue component, U or Cb, and the red component, V or Cr. This is more apparent in Fig. 21, where a picture is shown, along with the three YCbCr channels alongside the original image. The red in the image has a high red component value. So, red shows as near white in the Cr image. The dark blue jersey of the defensive tackle contains a mid level blue component. That jersey appears mid-gray in the blue component image. The human eye is more sensitive to the intensity information than the color information in an image. As a result, the color channels, Cb and Cr, are usually sub-sampled in order to compress the video with little psycho-visual loss.

The format of YUV 4:2:2, means that for every four gray-scale pixels, there are only two Cb and Cr pixels. Therefore, the size of the file is reduced to half of the original

size.

There are other reasons that the YCbCr color space is chosen in computer vision research. Changes in lighting conditions wreak havoc on computer vision algorithms. Color levels shift when the amount of illumination in the room changes. This forces thresholding levels and comparison images to change, usually causing the algorithm to fail. The YCbCr space is different from the RGB space in its susceptibility to lighting changes.

If illumination decreases, an image in the RGB space will see a decrease in levels in all three channels, red, green and blue, alike. In the YCbCr space, lighting usually only affects the Y component. The color-components are not altered. This makes the YCbCr space a very popular choice for vision algorithms. Although the lighting conditions can be carefully controlled in this instance, it is always desirable to be as robust as possible. For this reason, coupled with the fact that it is the native output of the cameras, the YCbCr color space was chosen for the testbed camera system.

Once the color space was chosen, the process of calibrating the cameras and threshold levels began. In order to ease the thresholding process, it would be nice if all cameras viewed the environment the same way. In other words, the color red would appear red in all four cameras. The color white would be consistent among all four cameras. This color consistency problem proved more difficult than it sounds, because each camera sensor has a variation built into it. This problem has recently attracted other researchers [8].

The process of setting the cameras and setting color threshold levels are closely linked. The process consisted of making changes to camera hardware settings and then examining the affects the hardware changes had on color levels. A standard print was made, shown in Fig. 22, consisting of true red, blue, green, black and white blocks. The cameras, already mounted in the ceiling, were each adjusted to a zoom of 3x. The print was placed in the camera's field of view on the testbed floor. The calibration print filled the view of the camera as seen in the figure. A frame of the calibration print was captured for each camera. Then, color histograms were calculated for each camera. The approach was

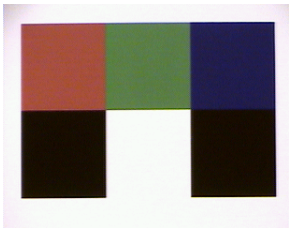

FIGURE 22 – Frame captured from a video camera of the image used to calibrate all cameras. Print consists of a block of pure red, blue, green, black and white.

to set the camera hardware so that color histograms from each camera matched as closely as possible. The three hardware settings mentioned earlier that affected the output the most were the aperture, red gain and blue gain. These three features also closely corresponded with the YCbCr color space. The aperture affected mainly the Y component. The red and blue gains primarily affected the Cr and Cb components respectively. There was some cross-correlation between the aperture setting and the color components. As the aperture decreased, allowing less light in, the color component histograms narrowed. This makes sense because as an object gets darker, less of its color information is conveyed. It is more difficult to identify the color of something in low light situations. This was, however, the worst of the coupling during calibration.

A benchmark camera was selected after iterative experiments of examining the color histograms and modifying camera settings. In order to quickly threshold colors, it is desirable that they be linearly separable. The camera hardware settings caused shifts in the color space, changing the way colors appeared, and in the end, classified. If there is a large separation between two colors in the color space, it will be very easy to classify the two of them correctly. As the separation decreases, the chance of misclassifying a pixel grows. In an attempt to minimize this, a hardware setting was found that separated the primary colors, red, green, blue, black and white, as much as possible. The histogram of this benchmark is shown in Fig. 23. This is a histogram of the camera looking at the calibration print. The job then became a process of modifying the other camera's hardware settings to produce a matching, or closely matching, histogram. This is similar to the approach taken in [8],

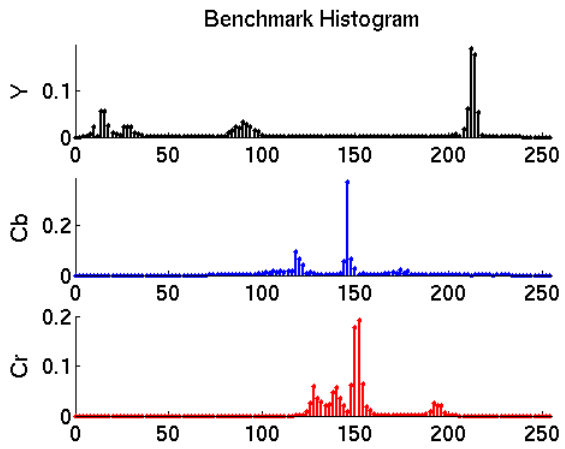

FIGURE 23 – Histogram of calibration print, taken from Camera 1, with desirable hardware settings. This provided the most separation between colors in the YCbCr color space.

with the exception that their process is automated. Automation requires more hardware to control the camera settings via software. Even with the automated approach, they conclude that there could never be a perfect match by merely adjusting hardware settings [8]. To achieve consistency, image processing correction is needed. The testbed camera system avoids any software correction to ensure fast tracking of the robots. The final histograms of the other three cameras are shown in Fig. 24. Notice how closely the histograms match.

C. Color Threshold Calibration

Once the proper adjustments were made to the camera hardware settings, the task was then to find proper threshold ranges for each of the critical colors: red, blue, green, black and white. These are the colors used in the hat patterns. Therefore, identifying them correctly is crucial.

Using the program shown in Appendix IV, a frame was captured from each camera while viewing the calibration print. Those frames were then processed interactively using the listing in Appendix V. The different color regions were identified. The mean, variance, minimum and maximum were calculated for each primary color in each of the color space channels, Y, Cb and Cr. The program also plotted histograms in three different

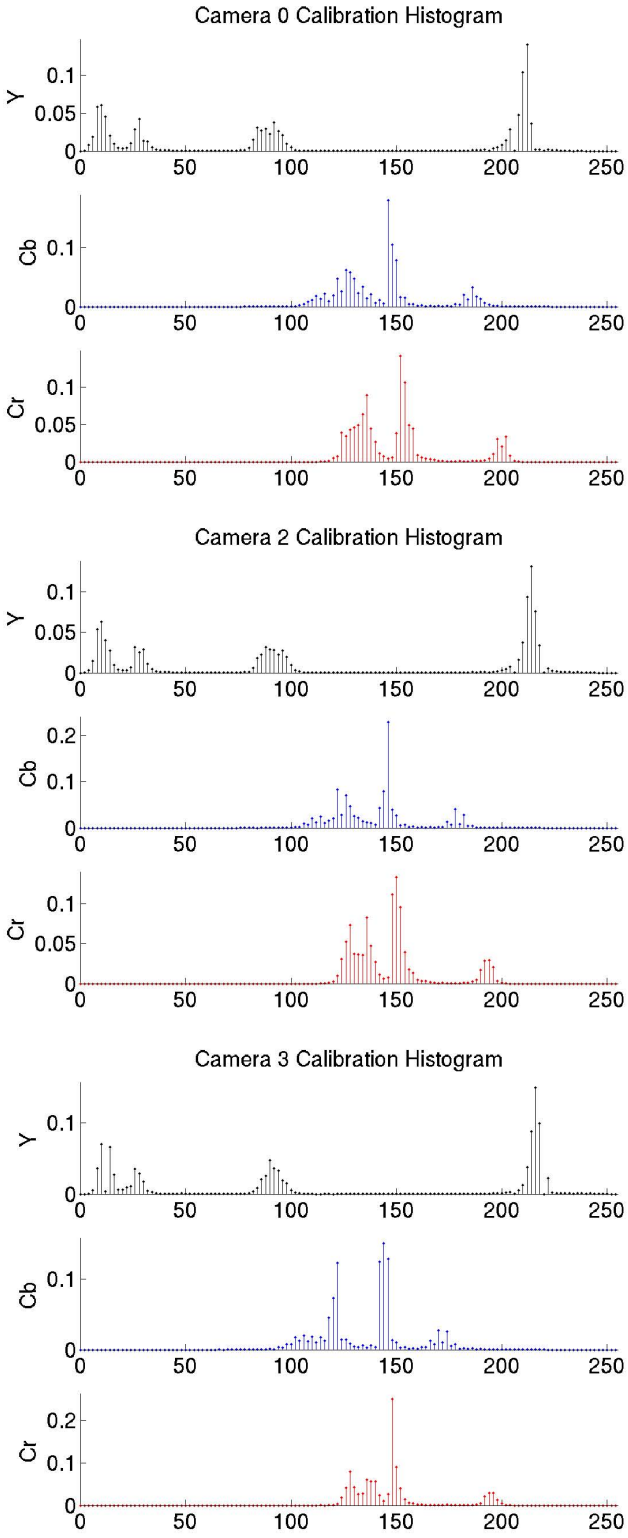

FIGURE 24 – Histogram of calibration print, taken from Cameras 0, 2 and 3, to match the benchmark of Camera 1. The histograms match very well with that shown in Fig. 23

.

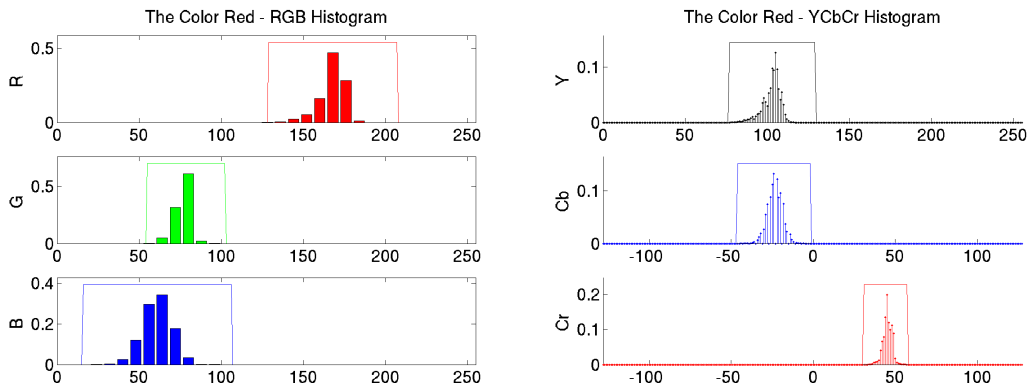

FIGURE 25 – Histograms of the color red in the RGB and YCbCr spaces, taken from Camera 0, in order to determine proper thresholding range for Red.

color spaces: the common RGB space, the HSI space, and the already-mentioned YCbCr space. In this way, thresholding could possibly be performed in any of these spaces. The resulting histograms from Camera 0 of the color red are shown in Fig. 25 in the RGB and YCbCr spaces. The RGB was not chosen as a color space for thresholding because of its susceptibility to lighting conditions as well as its need for conversion from the original format.

Another color space was investigated: the Hue-Saturation-Intensity (HSI) space. The HSI space was developed as a way to describe colors the way humans do. The first channel, Hue, is the color. The second channel, Saturation, describes the fullness, or tint, of the color. The final channel is meant to describe the intensity or gray-scale value. Unfortunately, the HSI space did not offer any advantages for color segmentation, such as large separations in different colors. The HSI space also requires a more complex, non-linear space conversion. This would be too costly in terms of speed to yield any great benefit. Thus, the native YCbCr format was chosen for thresholding.

The samples of red, green, blue, black and white were analyzed for each individual camera. Then, the samples from individual cameras were combined to give a global threshold range for each color. The results for the global thresholds for each color are shown in Fig. 26. Bounding boxes are shown around the histogram data to indicate a recommended

threshold range. In order to visualize the separation among all of the colors, the bounding boxes are plotted together in Fig. 27. The white range is shown as yellow on the plot. In order to easily separate two colors, they must differ in range in at least one channel. For example, red and green have some very similar characteristics. They overlap in the intensity channel and also have a similar blue content. However, in the red component channel, red is significantly higher. Red, for obvious reasons, has the highest red content of any other color, and by a significant amount. This is what makes the color red so easy to threshold without error. The tracking algorithm relies on the center color being classified correctly. That is why the center color for the multi-robot hat pattern was chosen to be red. The center color triggers the rest of the tracking algorithm. The color white is another example of a color that is easy to threshold. The separation of white from the other colors in the Y channel makes it an ideal choice for the flooring. It is, therefore, simple to separate a background pixel from a one belonging to a robot.

The threshold levels shown in Fig. 27 are those calculated by the calibration program as ± 3 standard deviations from the mean. According to the "*68-95-99.7 rule*" commonly used in statistics [12], this region of a normal distribution should encompass 99.7% of the pixels in each color. This gave a good starting point, however, the threshold levels had to be widened slightly for the testbed. This is due to the lighting changes as the robot moves around the testbed. The samples were taken from the center of each camera's view, where the calibration image was placed. The lighting tended to be brighter near the center of the testbed and slightly darker near the walls.

This broadening of the threshold ranges caused some problems for classifying black and blue. These two colors are closely related. The largest difference between the two of them is the blue component. However, the variance of the color blue in the blue component channel is fairly large. There is overlap in the blue component and the intensity channels, as well as the red component channels. This overlap leads to error in classification. This problem is overcome by carefully designing the hat patterns and search algorithms so that

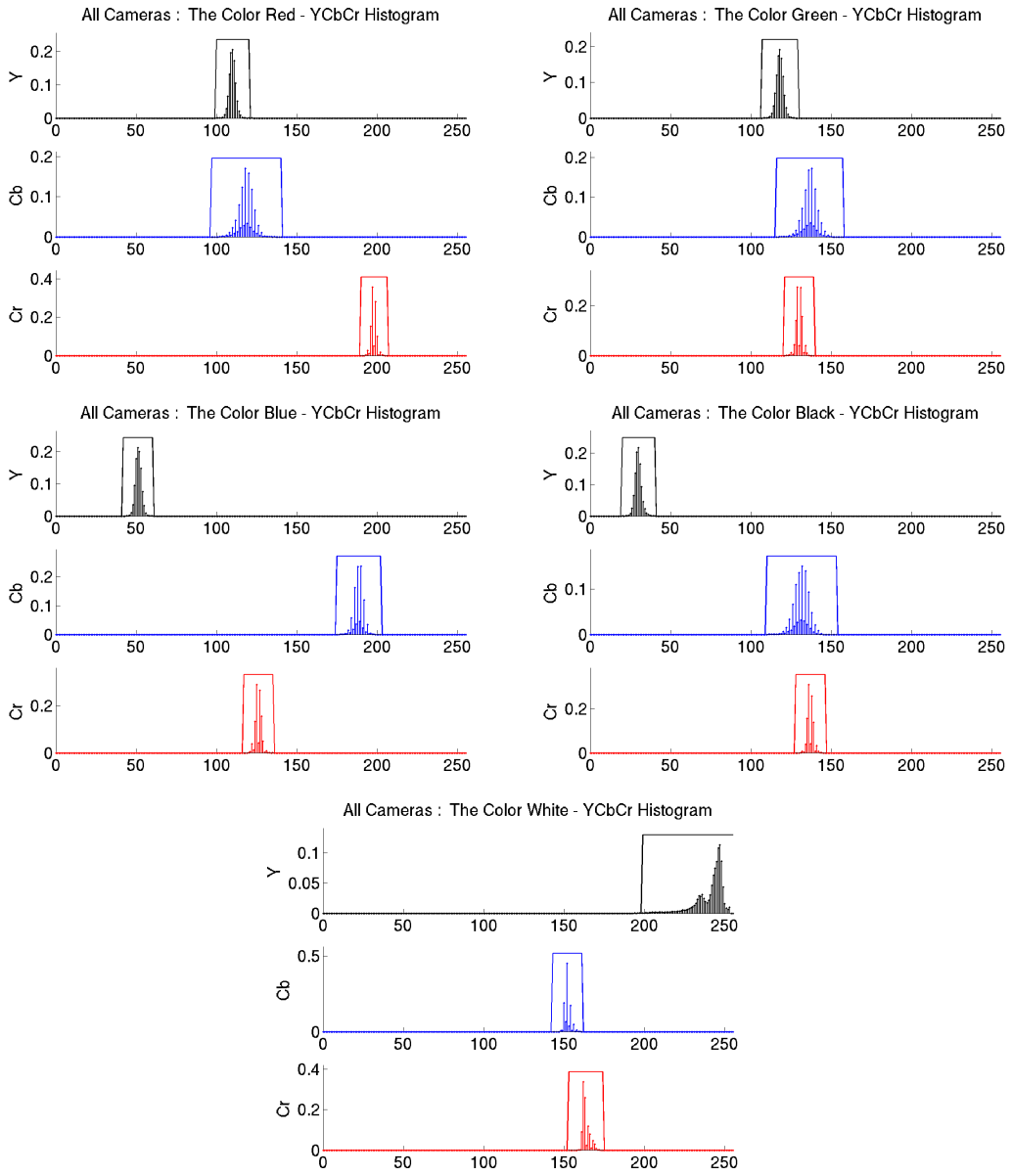

FIGURE 26 – From the Left: YCbCr Histograms of Red, Green, Blue, Black and White, taken from samples of all cameras, in order to determine proper thresholding range for the colors. The bounding boxes shown on the histograms indicate good thresholding ranges.

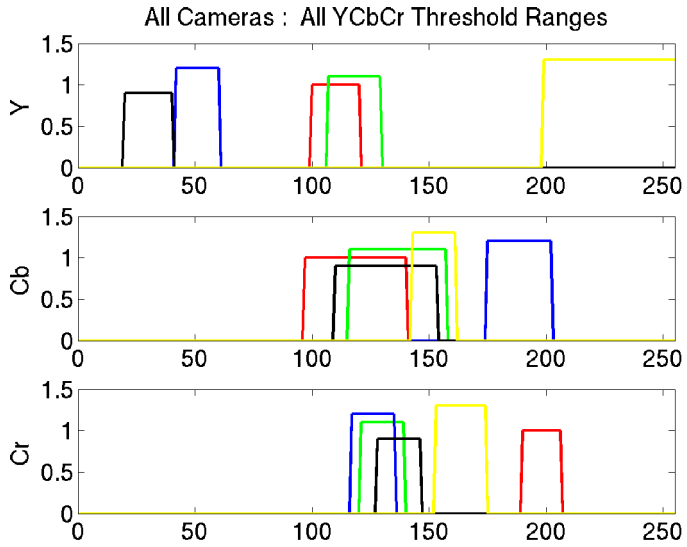

FIGURE 27 – Threshold ranges in the YCbCr space for the colors Red, Green, Blue, Black and White. White is shown as Yellow for visibility.

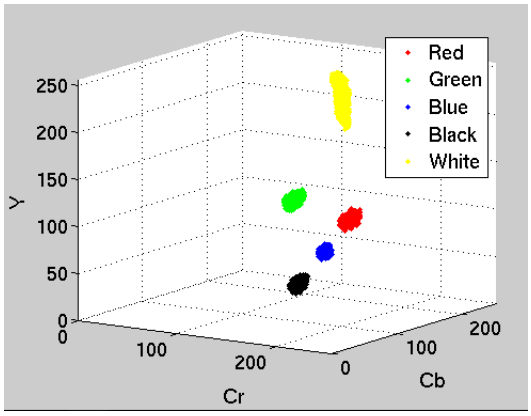

FIGURE 28 – Three dimensional plots of the colors Red, Green, Blue, Black and White in the YCbCr color space. White is shown as Yellow for visibility.

black and blue are never classified against each other. For the single robot algorithm, black is not even used. For the multi-robot algorithm, black is classified against white only. Blue is thresholded against green and white. For these two groups, the colors are linearly separable and therefore easy to classify with little error. The separability of the colors can also be seen in a three dimensional plot in Fig. 28, with each axis corresponding to the three color channels of the YCbCr space. Hyperplanes can easily classify these groups of

FIGURE 29 – Thresholded version of Fig. 8. This was thresholded in the YCbCr color space and turned out very well. The purple pixels represent ones that were not classified as any one of the primary colors: Red, Green, Blue, Black or White.

colors.

Figure 29 shows a properly thresholded image from the testbed. Notice this is the same image as in Fig. 8 in Chapter 3, except now, it has been thresholded. The thresholding performed here is only done for the primary colors of red, green, blue, black and white. There is another color present in this image and that is purple. Purple is used to show that a pixels was not classified as any other color. These non-classified pixels occur frequently in this image but are expected. They appear because of shadows cast, or the occasional view of the robot beneath the hat. Another place they crop up is surrounding the primary colors. This is due to one form of discretization error.

Figure 30 shows an example of the origins of this error. On the left is a continuous image of a circle, overlaid with a sampling grid. On the right, is a gross sampling of the continuous image. Each block of the grid, or sampling square, represents a pixel in a digital image. Notice how the curves of the circle are converted to blocked edges. This is one form of discretization error. The other is in the values of the pixels. Within each sampling square, the color level is averaged. The average color in each block, or pixel, becomes the level in the discrete image. Inside the circle, or outside the circle, the color level is rendered perfectly, because the levels inside these pixel blocks are uniform. However, on the edge of the circle, the color becomes something between the background, white, and the circle, black. This is exactly what happens when a digital camera samples a continuous world.

After sampling the continuous image, the boundary of the circle is neither black,

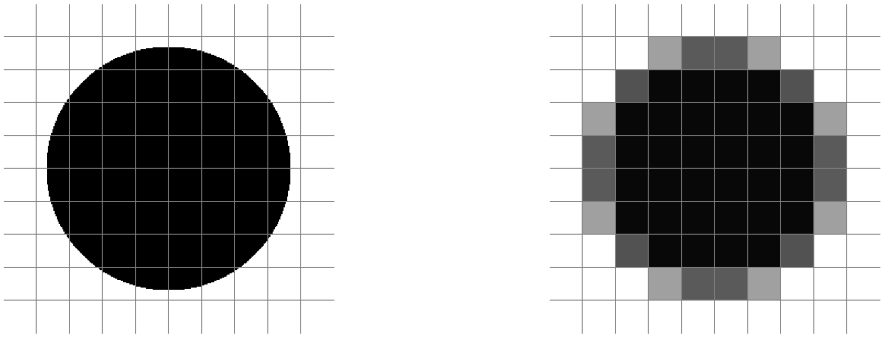

FIGURE 30 – Example of discretization error caused by sampling a continuous image. The left shows the continuous image of a circle with a sampling grid over the image. The right shows the low-resolution sampling of the continuous image.

nor white. If this boundary were thresholded with the method used by the testbed, the boundary pixels would not be classified as either black or white. The same is true for the pixels in Fig. 29. When any two colors meet, there is a *bleeding effect* caused by sampling the image. This bleeding effect can be seen in the boundary of the circle in Fig. 30. It was expected and accounted for in the design of the hat patterns.

The small black ring surrounding the red circle in the multi-robot hat is in place to cause uniform error between all of the robot hats. Immediately surrounding the black circle is a band which is unique to every robot. Different colors meeting on edge produce different bleeding effects. Some colors could average together with red to produce a color still in the red threshold range. This would cause a shift in the center of the pattern and be a source of error for calculating the position of each robot. The black ring acts as a constant for the bleeding effect, so that every multi-robot pattern experiences the same black-red averaging, regardless of the colors present in the robot number circle.

One issue with thresholding for the testbed is the cost of misclassification. The expense of classifying a blue pixel as green is quite high. That would mean that a robot would get mislabeled as having a different robot number. As a consequence, the pose data would never get sent to the correct robot. If a pixel is not classified as any of the primary

colors, the search will simply continue with the next available pixel. For the robot number search, there are many pixels within the quarter circle that can be tested. If the algorithm is unsure about a particular pixel, it is much better to look at another sample than to make a good guess.

For this reason, the threshold regions are purposely set small. If a pixel goes unclassified, therefore having a negative match, the algorithm still has a very high probability of success. On the other hand, expanding threshold ranges will lead to false positive matches, causing the algorithm to fail.

The most critical of all colors is red. It was chosen as the most important color because of its separability in the color space. The search algorithms perform the subsampled searches, keying on red. When a red pixel is discovered, some event is triggered, usually a detailed search. The red pixels are used to calculate the center of mass. If the threshold range is too narrow, the center of the robot could be shifted. A shift in the center of mass could cause the multi-robot algorithm to fail. On the contrary, if the threshold range is set too wide, other meaningless pixels could be mislabeled red, triggering a wasteful area search. There are stopgaps in place to recognize a false search or perform despite a shifted center; however, the balance between setting the red threshold range too large or small must be kept in check.

D. Position Calibration

Once the images are thoroughly thresholded and searched, the resulting position measurements are in units of pixels. In order for that information to be very useful to a robot, it needs to be converted to a unit such as meters. It is only a simple scaling factor. That scaling factor was found by measuring an object at the height of the robots in pixels and meters. The relation is a simple ratio. This was performed for both the vertical axis and horizontal axis of the camera. They were found to be only slightly different.

The origin of the testbed was selected as one corner of the room. From there, the

x and y axis extend in the positive direction. Typically, the axes of an image are defined as the positive x moving across the image and the positive y axis moving down the image. The origin is said to be in the upper left corner of the image.

The origin is still in the same place from image to testbed, but the axes in the images have been switched. This was done to keep with the standard definition of global axes for mobile robots. Had this not been done, the orientation angle, θ, would have been defined opposite of the norm, causing much confusion.

CHAPTER V
PERFORMANCE OF 2D TESTBED

This chapter examines the actual performance of the testbed. The original design goals will be studied for success or failure. Some of the goals could be assessed quickly, such as the proper utilization of space. The testbed was built in a near-square shape, allowing robots to move equally in all directions. Other goals required more study to prove success, namely, speed, accuracy, support of multiple robots, and proper communication while using a generic hardware platform.

A. Accuracy

The first of the goals examined is accuracy. It is very difficult to measure the accuracy of the camera system while the robot is in motion. At the millimeter precision, it is nearly impossible to reconcile the difference between the pose calculated by the vision system and the absolute position of the robot. In order to have some idea of the accuracy of the system, several measurements are taken while the robot is not moving [1]-[7]. Figure 31 shows a scatter plot along with histograms of 10,000 position measurements taken of a stationary robot using the single robot algorithm. The measurements of the orientation angle, θ, are represented in the histogram. From the figure, the algorithm performed quite well. Notice the scale of the axes are 10^{-3} meters. The position measurements were shifted to the origin for easier viewing. The farthest outliers are within 2 mm of the mean. The histogram shows that the θ measurements vary within $\pm 0.75^{\circ}$. This is an impressive feat and exceeds the original expectation of the orientation measurement.

The process was repeated for the multi-robot algorithm as well. Those results are

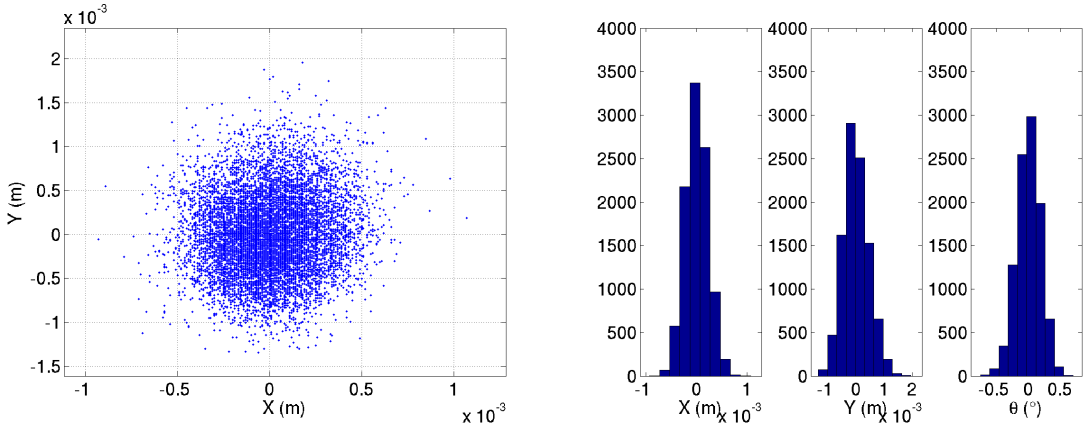

FIGURE 31 – Scatter Plot and Histogram of 10,000 measurements for a stationary robot using the single robot algorithm and hat pattern. Notice the scale of the axes is 10^{-3}.

shown in Fig. 32. This algorithm performed almost as well. The distribution was spread slightly more, with outliers of 2.1 mm from the mean. The orientation angle, θ, is shown in the histogram to vary $\pm 1^{\circ}$ from the mean. This variance is only slightly larger than the angle measurement from the single robot algorithm. This increase is a result of the loss of precision in the multi-robot algorithm. Remember that the single robot algorithm measures the orientation from an arctangent calculation. This gives a continuous distribution of values. The multi-robot algorithm measures θ by doing a circular search around the black/white semi-circle feature. The algorithm searches for one of the black/white edges. Therefore, this method is limited by the resolution of the image. Currently, the resolution of θ is limited to $\frac{1}{2}^{\circ}$ increments. These small variances present in both algorithms certainly meet the accuracy goals set for this testbed.

B. Speed

One major design goal for this testbed was real-time performance. The upper limit of the measurement frequency is set at 30 Hz by the frame rate of the overhead cameras. With the first implementation of the tracking algorithms, the measurement frequency was a sluggish 1.5 Hz, as discussed previously. This rate was unacceptable to be used as a

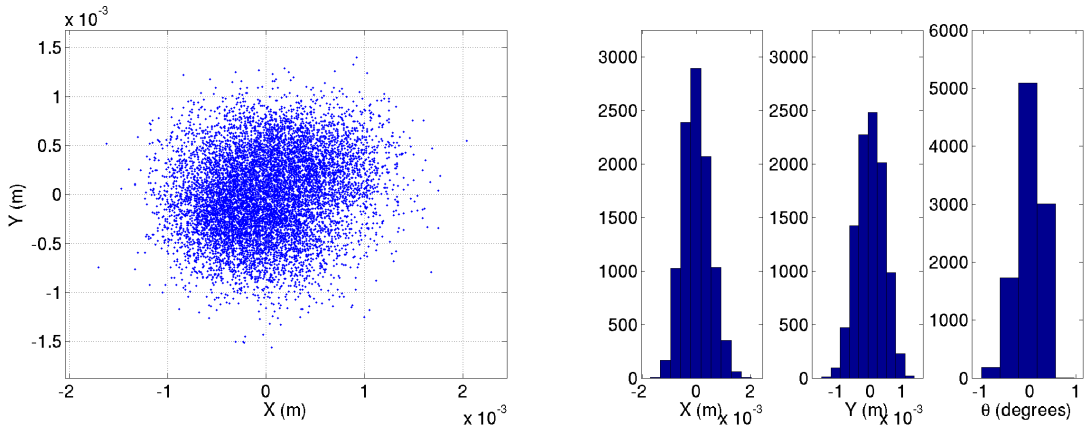

FIGURE 32 – Scatter Plot and Histogram of 10,000 measurements for a stationary robot using the multi-robot algorithm and hat pattern. Notice the scale of the axes is 10^{-3}.

real-time system.

After a major change of drivers, the frequency spiked to 30 Hz for both algorithms. Ten thousand measurements were taken and the experiment was timed. In 333.3 seconds, the 10,000 position measurements were made for the single robot algorithm. The process was repeated for the multi-robot algorithm using the three robots available. Even with three robots in the testbed, the measurement frequency performed at 30 Hz. This is certainly fast enough to meet the real-time goal for the testbed.

C. Other Goals

Given that the previous experiment was performed with three robots, it is apparent that the testbed is able to support multiple robots. Consequently, this design goal is satisfied as well. Also, during the experiment, the pose measurements were broadcast to the intended receivers using the UDP protocol described earlier. The pose messages were received free of error through the wireless 802.11g router. This success satisfies the communications requirement of the testbed. All of these experiments were performed using a single, normal-grade, desktop computer. No specialized digital image processing hardware was required to meet the measurement frequency goal of 30 Hz. This accomplishment sat-

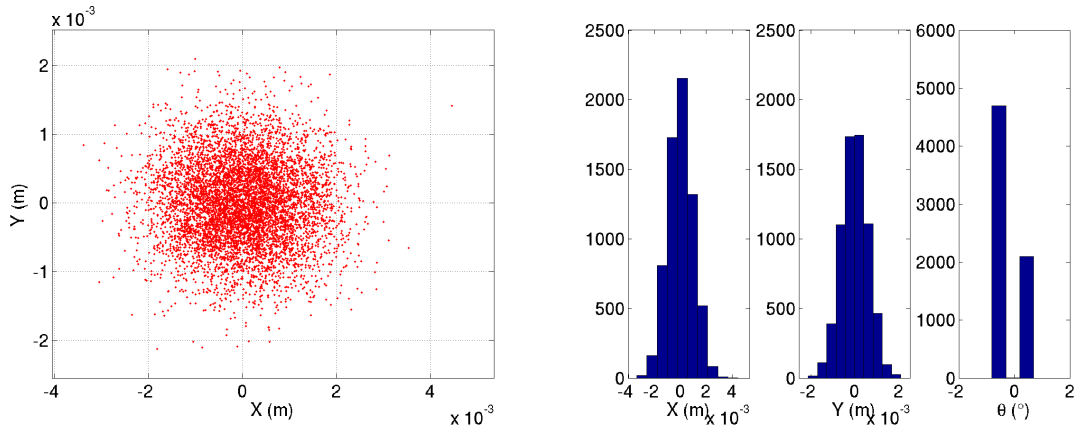

FIGURE 33 – Scatter Plot and Histogram of 10,000 measurements for a stationary robot using the multi-robot algorithm and hat pattern. To add noise to the system, some of the overhead lighting was shut off. Notice the scale of the axes is still 10^{-3}, though the variance has increased.

isfies the design goal of using a generic hardware platform to perform robot tracking. The tracking algorithm is implemented completely in software, making it highly flexible to the changing needs of robotics research.

D. Robustness to Noise

As an added measure of the performance of the testbed, noise was added to the environment by changing the lighting conditions. Changes in lighting conditions are a considerable source of failure for computer vision algorithms. It is used here to the measure the robustness of the system.

After switching off several lights in the laboratory, the accuracy experiment was performed again. Figure 33 shows the results of those measurements in a scatter plot of the position, accompanied by histograms of the pose. These measurements were taken using the multi-robot algorithm. It is the more complex of the algorithms and is therefore, more likely to fail. The scale of millimeters did not change in the scatter plot, but the variance approximately doubled with the noise. Accuracies could be reported as ± 4 mm here. The orientation angle, θ, experienced a little less trouble, still showing a range within 1° of the mean. Although, it should be noted that the histogram of θ shows a decrease in resolution.

The step size made an apparent increase to 1° from $\frac{1}{2}^\circ$.

This experiment was repeated for extremely low illumination, but the algorithm failed completely. It would not acknowledge that there were any robots in the testbed. The good news is that if there is some noise present, and the algorithm is tracking robots, the error margin will be only slightly increased. If the noise is too large, it will be apparent by not tracking at all, rather than yielding even larger error margins.

CHAPTER VI
MULTIPLE MOBILE ROBOTICS PLATFORM

A. Hardware

In order to perform experiments and give position feedback in real-time, a capable hardware platform is used. The four cameras overhead are Sony UC5300 auto-focus, zoom cameras with a 1/4 inch high resolution CCD sensor [13]. The video data feeds into a Video-4-Linux MDVA3000 frame-grabber [14], which is capable of capturing eight full-rate NTSC video signals, at a resolution of 640x480, at 30 frames per second. An HP Pavilion a1430n desktop computer with an 64-bit AMD Athlon X2 dual core 1.0 GHz processor and 2.0 G-bytes of RAM performs the image processing to track multiple robots. The wireless communications with the mobile robots is done through a standard Linksys 802.11g router.

The robots themselves are Evolution Robotics ER1's [15]. The ER1 is a highly flexible unit that allows the physical structure of the robot to be designed and built by the user. The robot is made of various beams and connectors that can be used to create a variety of structures. Currently, there are three ER1's in the lab, all of which have a different shape and size. The robots are shown in Fig. 34. One robot is in the standard Evolution Robotics configuration. The second robot is near the same size, but built differently to allow easier access to the driver hardware than the standard design. The third robot is built with a hitch and trailer, allowing for some interesting movement experiments to be performed. One other advantage of this robot, besides the flexibility of the body style, is the cost. A single ER1 only costs $300. Most mobile robotic platforms are near the $2,000 range, and are not

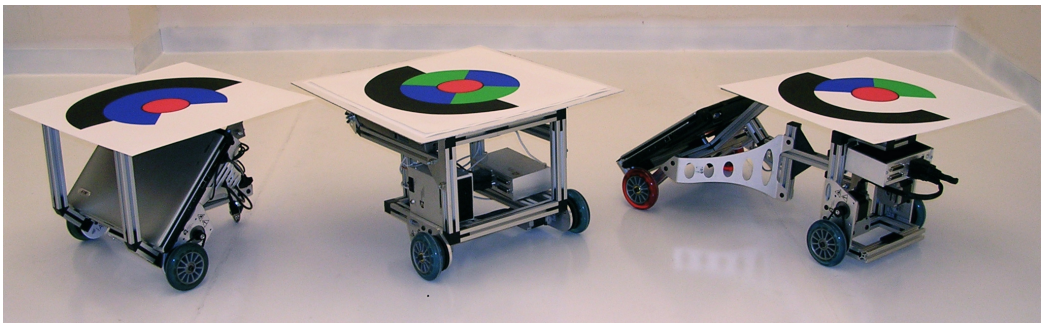

FIGURE 34 – Picture of 3 Evolution Robotics ER1's. The robot bodies can be reconfigured to build a variety of shapes. The standard configuration is on the left. The robot to the right is the hitch and trailer style. All of these robots are shown wearing their "hats", for tracking by the camera system.

nearly as flexible. The ER1 hardware is driven by a laptop computer, which interfaces via Universal Serial Bus (USB). The control box also connects to two powerful stepper motors as well as other input/output (I/O) devices such as infrared (IR) sensors and a gripper arm. There are also ports for analog inputs and digital inputs and outputs for custom sensors and devices.

The ability to control the robot with a laptop is a distinct advantage over other robots. Several popular robots are driven by embedded processors with limited processing speeds and memory availability. A laptop computer offers enhancements in both of these areas and also interfaces more easily with other devices such as cameras or range finding sensors.

The laptop computers that drive these ER1's consist of a couple of models. The first is a Dell Latitude 100L with a 32-bit 1.0 GHz Intel Celeron processor and 512 Mb of RAM. The other two laptops are Acer Aspire 5003WLCi's with 64-bit 1.8 GHz AMD Turion processors and 512 Mb of RAM. All of the laptops are equipped with 802.11g wireless cards to communicate with the camera system and each other.

B. Software

Though the hardware of the ER1 is fairly robust and highly flexible, the software

accompanying the robot is not. Unfortunately, the ER1 was originally designed as an educational tool for middle school or high school students with no prior knowledge of robotics or programming. The control software consists of a graphical user interface (GUI) which allows a user to setup a chain of prior developed behaviors. The user can link together several behaviors to get an emergent behavior. This system is very easy to learn, but highly inflexible. It was not suited at all for research use in this laboratory.

Evolution sells a more advanced control software for the ER1 which provides the flexibility needed for research. However, with a price of $3,500 per license, other suitable options were explored. An open-source project called Player/Stage was found to meet the research need.

Player/Stage [16] was developed at the University of Southern California as a robotics software platform. Player is a robot server which provides a hardware abstraction to physical robots and sensors. Drivers written for Player, allow Player to connect to many different robotic hardware platforms and sensing devices. Client programs can then be written to communicate to Player, without worrying about the physical robot. Player then communicates to the hardware, operating in the Client-Server model.

Stage is a two-dimensional robot simulator that communicates with Player. Stage can simulate hundreds of robots simultaneously. Robots can be built in Stage with a myriad of sensors and interact with user-created worlds. With the Player abstraction layer, client programs can be written and tested on simulated robots as well as actual robots. Often, the client does not know whether it is connected to Stage or a physical robot.

Currently, Player utilizes TCP socket communications. So, any language that is able to communicate via TCP can be used to write Player client programs. There are Player libraries written in C, C++, Java, and Python, as well as several others.

Player/Stage is one of the most widely used robotics platforms among researchers in the U.S. and the world. Since Player/Stage is open source, it is available free of charge. The combination of its popularity, open-source availability and language flexibility make

Player/Stage a powerful software platform for robotics research.

Player/Stage was built for Linux or Unix-like operating systems. That required the laptops to run some Linux distribution. Mandriva 2006 was chosen after considering several others, mainly for its free cost and local knowledge base. Mandriva is an offshoot of the popular Mandrake Linux distribution. It is also the most widely used Linux distribution in the Electrical Engineering Department at UofL. That made it the best choice for use in the robotics laboratory.

One anecdote worth mentioning is that the Linux kernel needed to be modified in order to communicate with the ER1 hardware. Evolution Robotics applied for unique product and vendor ID's for their USB connection. So, in order for Linux to recognize the hardware, those ID's needed to be added to the current list in the kernel. These changes are listed in Appendix VI.

CHAPTER VII
APPLICATIONS

Equipped with a tool to quantify movements of mobile robots, research can be performed in many areas. There are many applications that can be studied. Anything from coordinated movement, to path planning, to obstacle avoidance can be examined with this testbed. To begin, some control experiments were conducted.

A. Smooth Path Control

First, a controller was developed to move a differential drive robot along a smooth path from any point A to point B. A differential drive robot is one that uses two drive wheels, controlled independently by two motors. The wheel planes are parallel with each other. This constrains the movement of the side, or local y, direction to zero. In mobile robotics, this is known as a non-holonomic constraint. This is why differential drive robots are also referred to as a type of non-holonomic robot. Its motion is restricted to the X_R direction, as shown in Fig. 35. Holonomic robots are free to move in any direction at any given time. For this reason, they are typically called omnidirectional robots. For an omnidirectional robot, the task of moving from point A to B is very simple. Without any constraints, these robots simply move in a straight line directly to the destination. The orientation can even be changed in motion without affecting the path. A differential drive robot presents more of a challenge. The kinematic constraints introduced prevent the robot from moving directly toward the goal.

One way of moving from point to point for the differential drive robot, is to simply rotate toward the goal, move in a straight path to the goal and then rotate to the desired

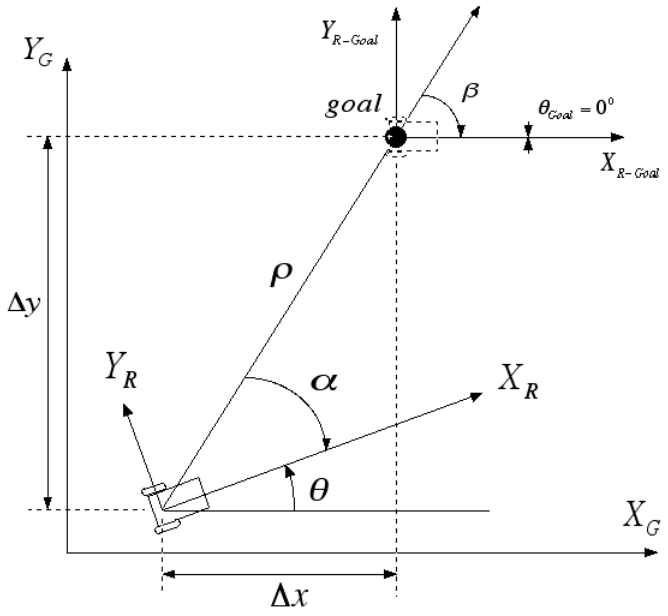

FIGURE 35 – Diagram of the robot's frames of interest. The control variables ρ, α, and β are illustrated. The goal here is shown in the upper-right, with the orientation aligned with the global axes.

orientation. However, this approach yields discontinuities in the actual movement. This may be acceptable for small ground vehicles, but the ultimate goal of this research is to later apply these methods to actual systems. Physical systems such as automobiles or airplanes have even more constraints. Automobiles cannot rotate in place like the differential drive robot. Airplanes require minimum velocities in order to provide lift and minimum radii to turn. Most cannot hover in place.

A better approach is to find a way to control a smooth path to a specified target. This can be expressed as finding control of $v(t)$, the linear velocity, and $\omega(t)$, the angular velocity, so that the error in current position and desired position is driven to zero.

$$\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = K \cdot e = K \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{Global\ Error} \tag{7}$$

The kinematics of a differential drive robot can be described by Eqn. 8, where $\dot{x}$ and $\dot{y}$ are

the linear velocities in the global reference frame.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_G = \begin{bmatrix} cos\ \theta & 0 \\ sin\ \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{8}$$

Next, consider the coordinate transformation into polar coordinates. Δx and Δy are shown in Fig. 35.

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{9}$$

$$\alpha = -\theta + atan2(\Delta y, \Delta x) \tag{10}$$

$$\beta = -\theta - \alpha + \theta_{Goal} \tag{11}$$

Note that $atan2$ is the four quadrant inverse tangent. This new coordinate system is shown in Fig. 35. Using the new coordinate system described by (9)-(11), Eqn. 8 is transformed to give a new description of the system.

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -cos\ \alpha & 0 \\ \frac{sin\ \theta}{\rho} & -1 \\ -\frac{sin\ \alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{12}$$

Starting at some arbitrary location, (ρ_0, α_0, β_0), the task becomes driving the error between the starting and final location to zero. At a minimum, this can be achieved using proportional control of the three variables: ρ, α, and β

$$v = k_\rho \rho \tag{13}$$

$$\omega = k_\alpha \alpha + k_\beta \beta \tag{14}$$

where k_ρ, k_α and k_β are the proportional control constants for ρ, α and β, respectively. Substituting the control law of (13) and (14) into Eqn. 12, the system is then described by

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -k_\rho \rho cos\ \alpha \\ k_\rho sin\ \alpha - k_\alpha \alpha - k_\beta \beta \\ -k_\rho sin\ \alpha \end{bmatrix} \tag{15}$$

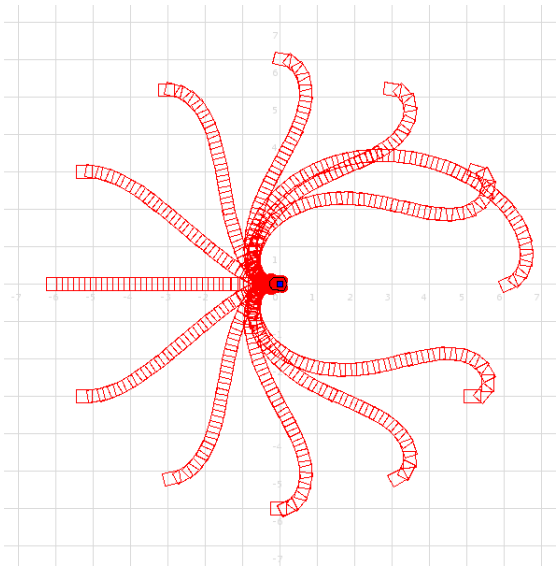

FIGURE 36 – Stage simulation of the smooth trajectory controller. The robot was placed at different poses on a 6 m radius circle. The starting orientation angle was always 0°. The robot found a smooth path to the origin each time.

More details for this derivation can be found in [17]. This system will drive the robot along a smooth trajectory to the goal pose. It should be noted that the sign of v is constant. Therefore, the direction the linear velocity of the robot is restricted to either forward or backward. Forward was chosen for the design of this controller. This decision affects another area of the controller. The angles α and β are defined using the vector $\vec{\rho}$. As the robot approached the goal, $\vec{\rho}$ gets smaller and smaller in magnitude. If the robot passes the intended goal, this vector remains small in magnitude, but suddenly changes almost 180° in direction. This causes the small errors in α and β to spike to almost 180° as well. These large errors then provide a large input to the angular velocity. In other words, since the linear velocity is constrained to forward motion, the robot attempts to circle around and make another pass at the goal pose. This situation is corrected by monitoring the derivatives of the angles, $\dot{\alpha}$ and $\dot{\beta}$. Whenever the derivatives spike in value, the controller is stopped.

The experiment was first performed in the simulator Stage, as seen in Fig. 36. The robot was placed at different locations around a 6 m radius circle and commanded to move to the origin. The orientation at each starting location was 0°. The controller was then tested

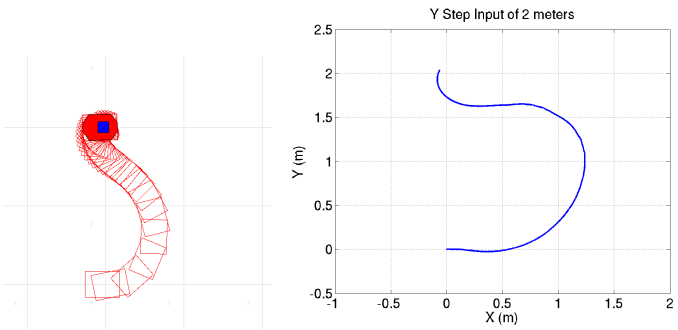

FIGURE 37 – Stage simulation and actual experiment of the smooth trajectory controller. The command was to move from the (0,0,0°) to (0,2,0°). The robot moved along a smooth path in the simulator and the hardware. The hardware shows a small steady-state error though.

with the ER1 robot. Figure 37 shows the path taken by the ER1 when given a command to move from (0m, 0m, 0°) to (0m, 2m, 0°). This figure shows the results in the simulator and the hardware. Notice that there is a small steady state error in the final position with the ER1. It is infeasible to reach any given position exactly. There is a limit to the accuracy of the position measurement. There is also a small time delay between measurements and movements. The actual controller is implemented using a threshold within a given error margin to decide when the goal pose has been reached. When the robot arrives within the acceptable threshold range, the controller stops.

B. Circular Path Control

The second application was the control of a differential drive robot along a circular path. All of the hardware experiments were performed in Stage first. Stage serves as a useful tool to quickly prototype an algorithm without the potential of damaging physical robot hardware. There are also very few changes that need to be made to the program when moving from the simulator to actual hardware.

1. Simulation

In the initial controller design, the radius of the circle was tracked. A constant linear velocity was maintained, while the error between the desired radius and the current radius proportionally controlled the angular velocity of the vehicle. The results of this method are shown in Figure 38. These images are taken from Stage at different intervals of the simulation. The robot is shown as the solid red polygon. The red trails show the path taken by the robot for illustration. In this simulation, the robot attempts to track a circle of radius 3 m in the counter-clockwise direction with a constant linear velocity of 0.2 m/s. The robot is given a starting location directly on the desired radius. As Fig. 38 shows, this controller is unstable. The oscillations start to occur during the first pass of the circle. They grow in the second pass. The robot does not even finish the third trip around the circle. The oscillations grow so much, that the robot actually moves in the opposite intended direction as shown in the last frame.

This experiment was performed for several different proportional gains, but the results were the same. Simply changing the proportional gains did not fix this problem. For the non-holonomic robot, this controller design was inherently unstable. Some other control variable was needed. As several others have done, the tangent angle was added to the feedback loop [18], [19]. Now, the robot not only tracked the radius of the circle, but also the angle tangent to the circle at the robot's position. Equation 17 gives the new control law, where ρ is the radius error and τ is the error between the current orientation and the tangent of the circle. This brought stability to the system.

$$\tau = \theta - \theta_{tangent} \tag{16}$$

$$\omega = k_\rho \rho - k_\tau \tau \tag{17}$$

Figure 39 shows the results of the experiment with the added angular control. The experiment was the same, with the exception of the starting location. This controller performed so well, that robot could be positioned at different poses and successfully track the circle.

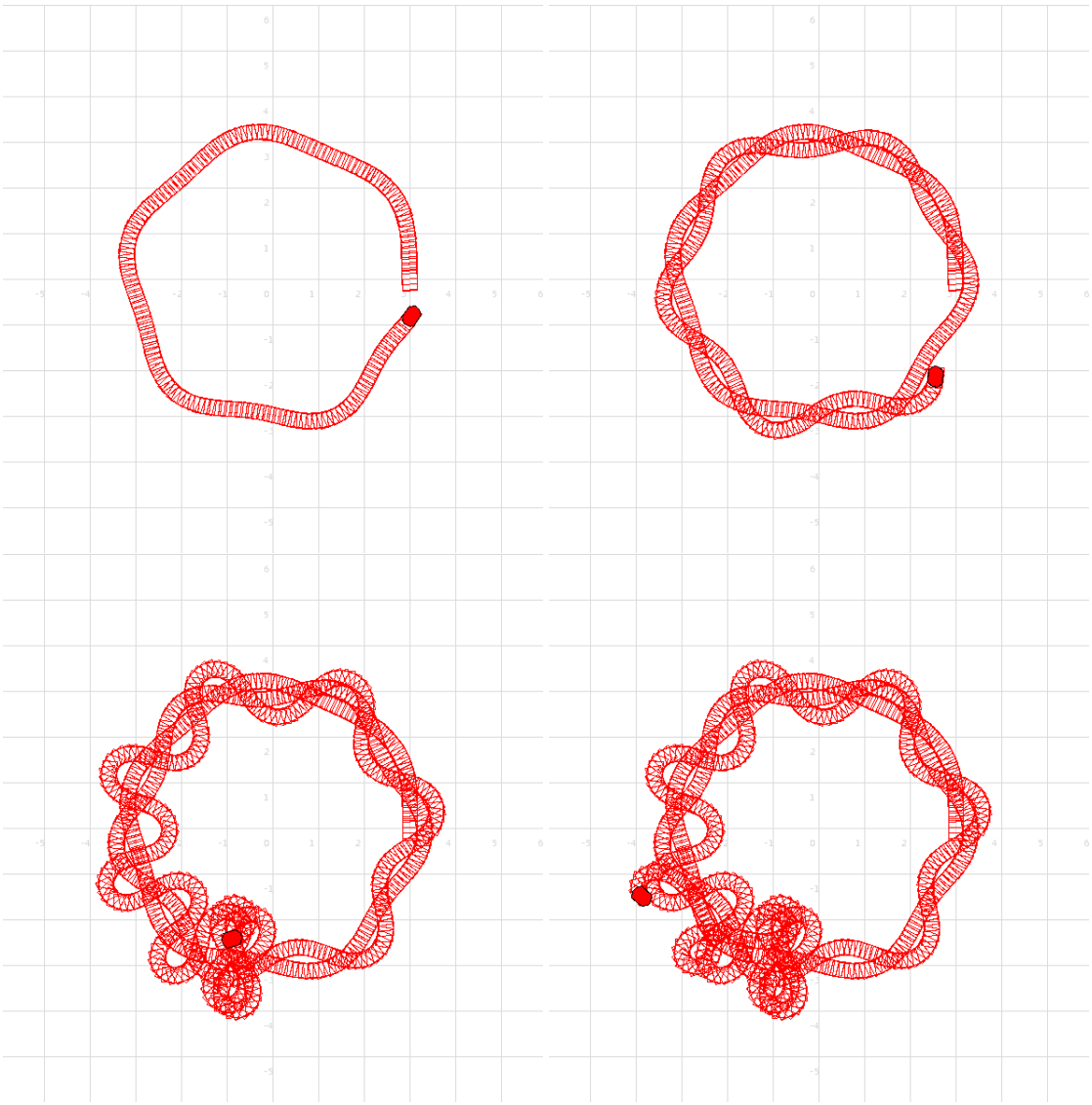

FIGURE 38 – Simulation of circular path controller, tracking the radius at 3 meters, with a forward speed of 0.2 m/s. The starting point was (3.0, 0, 90°). As time progressed, the oscillations grew until the algorithm failed.

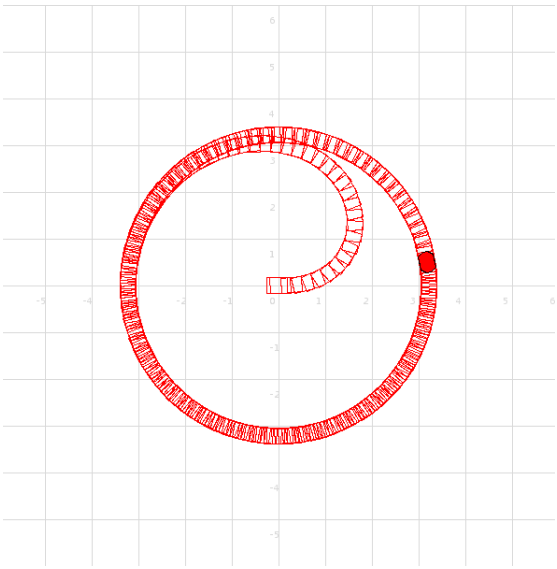

FIGURE 39 – Simulation of circular path controller, tracking the radius of 3 meters and the tangent angle of the circle, with a forward speed of 0.2 m/s. Even when the starting location was the origin, this controller performed very well.

Here it was started at the origin. This frame was taken after three passes around the circle. One thing to note in Fig. 39 is the steady state error present in this controller. The center of the robot should pass directly over the 3 m radius. In the figure, the inside wheel of the robot is actually just outside the 3 m radius. This could be corrected by adding integral control to the feedback loop.

2. Single Robot: Odometry and Camera System Feedback

Once the control algorithms performed well in the simulator, they were applied with the physical robots in the testbed. In the simulator, there is only one type of pose feedback. It consists of the "Eye of God" perfect measurement. With the robot hardware, there is not the benefit of perfect measurement.

The ER1's are equipped with internal odometry to keep track of their pose. With the testbed, there is also the possibility of receiving feedback through the vision system. Both of these systems were examined during these experiments.

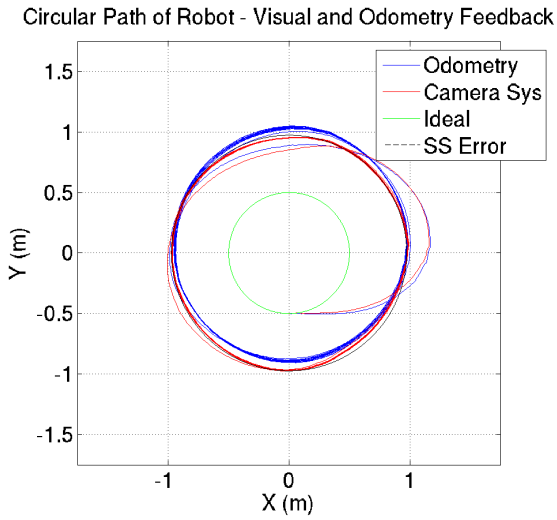

FIGURE 40 – Experiment of circular path controller in testbed. Tracking is done for a 0.5 radius and forward speed of 0.2 m/s. Plot shows the experiment with odometry feedback in blue, and camera system feedback in red. There is a steady-state error present in this controller.

Figure 40 shows the results of the hardware experiments. The robot was given a circle of radius 0.5 m to track with a constant linear speed of 0.2 m/s. For these experiments, the robot started directly on the circle to track. The figure shows the results of the experiment using both the internal odometry as feedback and the overhead camera system as feedback. The steady state error mentioned previously is more apparent here than in the simulations. The error draws to half a meter.

The vision system proves to be more accurate than the odometers. There is an apparent shift in the odometer's sense of the origin as it makes its first pass around the circle. This is a well-documented error found in odometers. The point of origin suffers from a "drift" as the robot moves. After the shift, the odometers are fairly consistent, but there is a larger variance as the robot continues to track the circle. This can be seen from the spread of the blue lines in various sections of the path. As the robot moves, the odometry uncertainty grows. The variance in the vision system is much smaller. Of course, it keeps track of the origin as well.

Since the orientation angle, θ, was also measured during the tracking experiment, that information was plotted in Fig. 41. This is a representation of the movements of the

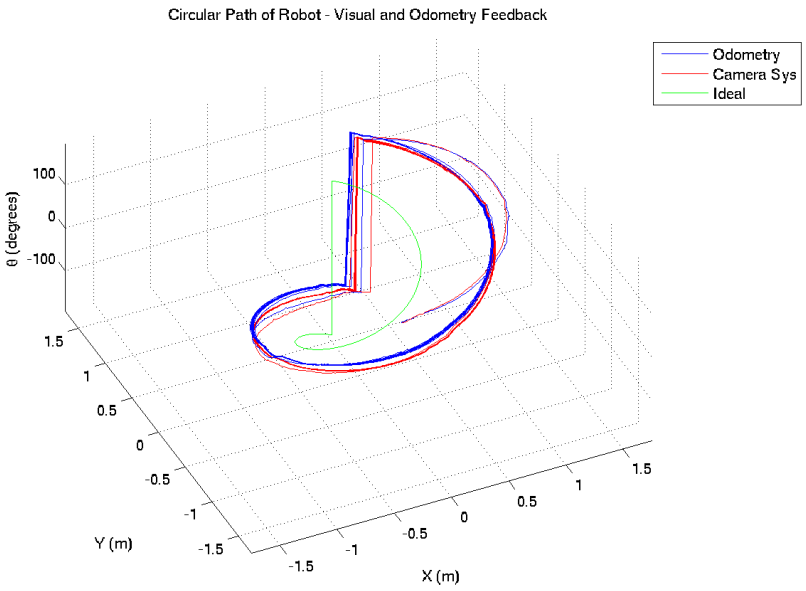

FIGURE 41 – Experiment of circular path controller in testbed. Tracking is done for a 0.5 radius and forward speed of 0.2 m/s. Plot shows the experiment with odometry feedback in blue, and camera system feedback in red in three dimensions to view the change in θ. The discontinuity is expected as θ moves from 180° to -180°.

robot in three dimensions, with the vertical axis representing θ. The sharp discontinuity is expected as the robot moves around the circle from 180° to -180°.

One interesting feature present in this diagram, not represented in the previous, is the consistent "ripples" in the path. These are very small but repeated shifts in θ. These can be explained by the seams in the testbed. As the robot hits the seams of the floor, there are small bumps that cause the robot to change orientation slightly. Unfortunately, the seams could not be constructed perfectly, but this effect is another testimony of the accuracy of the overhead camera system.

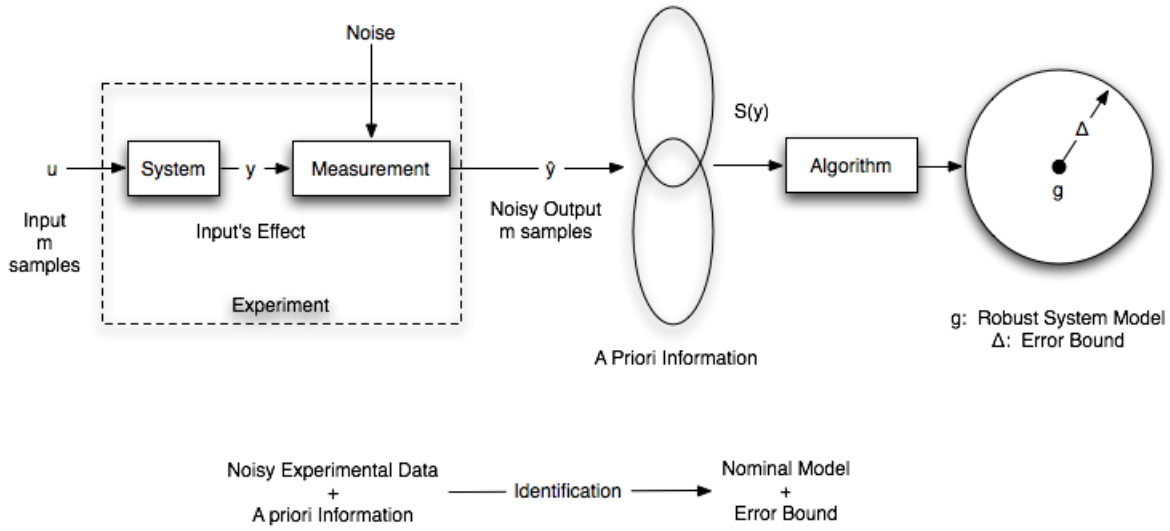

FIGURE 42 – A summary of the Robust Identification Framework.

C. Robust Identification of a Single Robot

Given noisy experimental data and some a priori information about the class of models (*not* the order of the model) robust identification procedures generate a nominal model and bounds on the identification error as shown in Fig. 42. The available a priori information consists of a lower bound on the relative stability margin of the plant, an upper bound on a certain gain associated with the plant, and an upper bound on the noise level. More detail on robust identification can be found in [20], [21], and [22]. One of the major differences and advantages of the robust identification algorithm compared to other classical modeling algorithms [21] such as maximum likelihood estimation, least squares estimation, etc., is that it is deterministic rather than stochastic. Another advantage is that it takes into account the measurement noise, and finally, it does not assume a pre-fixed mathematical structure as other classical methods do.

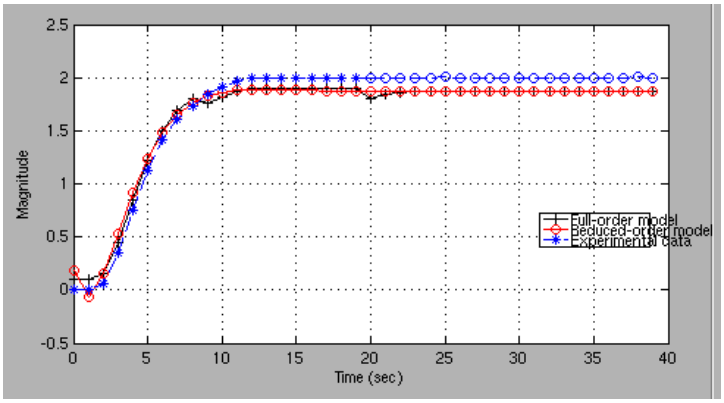

FIGURE 43 – Result of robust identification using a 2 meter step input. The identification was done using the internal odometers as feedback. The full order and reduced order models are shown in black and red, respectively. The actual output is shown in blue. The reduced order model was reduced from 25^{th} order model to a 2^{nd} order model.

1. Modeling with Odometry Feedback

In order to model one of the ER1 robots, a step input was supplied to the robot. The step input was given as a change in position of 2 meters in the forward direction. The resulting output was then measured. Both the output and input were supplied to the robust identification algorithm. This process was performed twice. The first model generated was done using the robot's internal odometers as position feedback. The second experiment was done using the overhead vision system. Figure 43 shows the result of the time domain experiment using the odometers as feedback. The blue points are the experimental data. The black and red data points show the full order and reduced order models generated from the robust identification. In this figure, "*" denotes experimental data points used in the identification algorithm, while "$\circ$" indicates the points not used in the identification algorithm. The step response of the model matches the experimental time-domain data very well and is within the maximum error bound.

To find the upper error bound of the system, the position of the robot was measured by changing the lighting conditions. This added noise caused variation in the testbed's measurement of the robots. The maximum error at the time was measured at 0.1 meters.

These models were found using ℓ_1 *identification*, which is strictly based upon exper-

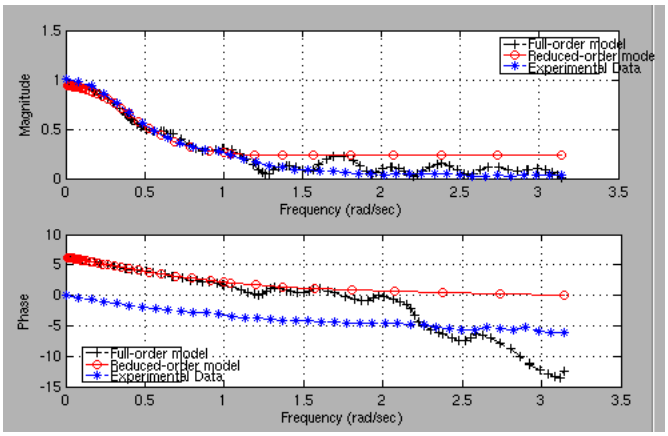

FIGURE 44 – Robust model comparison in the frequency domain. The frequency data is shown for the actual data, the full order model and the reduced model. The model matches the data well in the time and frequency domains.

imental time domain data. Another form of robust identification, known as H_∞ *identification* uses frequency domain experimental data to calculate a model. There is another hybrid method, *mixed* ℓ_1/H_∞ *identification* which utilizes time domain data as well as frequency domain data. Figure 44 shows the frequency response of the experimental data as well as the responses of the full and reduced order models for comparison. This figure shows that the model found matches well in the frequency domain. Thus, H_∞ or mixed identification is not needed. The model produced by ℓ_1 identification is a discrete model represented in the state space. The typical system is usually defined as:

$$x[n+1] = \mathbf{A}\,x[n] + \mathbf{B}\,u[n] \tag{18}$$

$$y[n] = \mathbf{C}\,x[n] + \mathbf{D}\,u[n] \tag{19}$$

The robust identification yielded the following matrices for the reduced, second order system:

$$\mathbf{A} = \begin{bmatrix} 0.8591 & -0.1770 \\ 1 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{20}$$

$$\mathbf{C} = \begin{bmatrix} 0.0101 & 0.2428 \end{bmatrix} \quad \mathbf{D} = [\,0.0548\,] \tag{21}$$

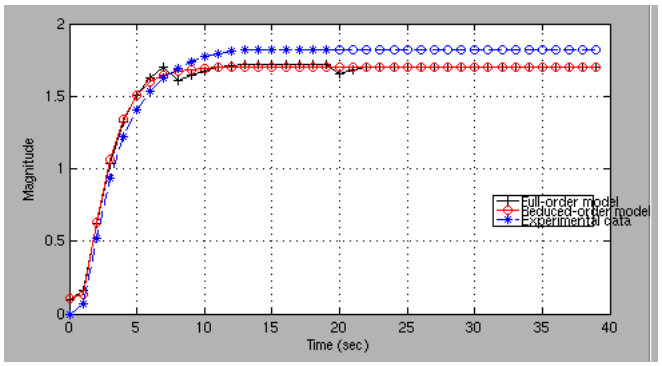

FIGURE 45 – Result of robust identification using a 2 meter step input. The modeling was done using the camera system as feedback. The full order and reduced order models are shown in black and red, respectively. The actual output is shown in blue. The reduced order model was reduced from 25th order model to a 2nd order model.

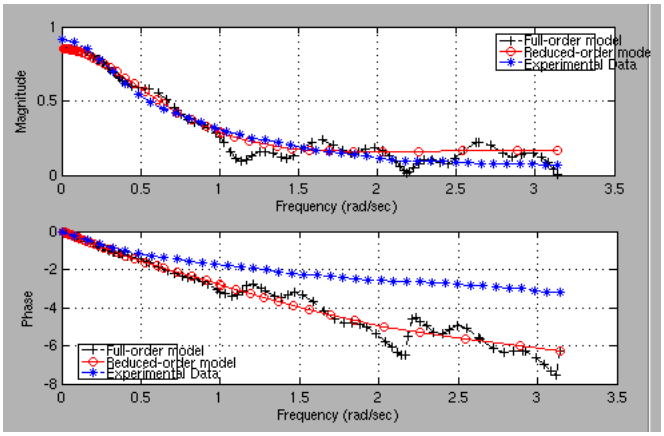

FIGURE 46 – Robust model comparison in the frequency domain. The frequency data is shown for the actual data, the full order model and the reduced model. The model matches the data well in the time and frequency domains.

The same experiment was performed using the testbed overhead vision system for the position feedback during the step input experiment. Figures 45 and 46 show the results of the ℓ_1 identification. The models match well in both the time domain as well as the frequency domain. The matrices to define the reduced, second order model are given in Equations 22 and 23.

$$\mathbf{A} = \begin{bmatrix} 1.2637 & -0.4291 \\ 1 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{22}$$

$$\mathbf{C} = \begin{bmatrix} -0.1240 & 0.2635 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 0.0925 \end{bmatrix} \tag{23}$$

D. Leader-Follow Experiments

Currently, considerable research is being done with mobile robot coordination. These studies cover areas such as communication, distributed sensing, and environment mapping, to name a few. Another area of this research is group movement or formation movement. The formation studied here is a leader-follow formation.

To breech this topic, the smooth path controller was extended to accept a constantly varying goal state. A leader communicates its location to the follower. The follower receives the leader's position and sets the goal state at a specified distance behind the leader. The error between the follower's current position and newly defined goal state is fed to the smooth path controller. This entire process repeats as the leader moves. The leader continues to send its pose to the follower. The follower updates the goal based on the new location of the leader and calculates a new error. Both the leader and follower receive their own pose information from the overhead vision system of the testbed.

It is easy for a follower to track a leader if the follower knows the final goal of the leader. The problem of following a leader is then reduced to the follower finding its own way to the goal. This is not really following at all, but merely two independent robots moving to some final state. In the method presented here, the follower robot is unaware of the destination of the leader. The follower's task is to simply trail the leader based on current information from the leader robot.

Another thing to note is that the leader-follow algorithms use decentralized control. The robots communicate with each other to exchange pose information, but the movement control of each robot is performed by each robot. There is not a central controller that gives movement commands to the robots in the testbed. Decentralized control presents a greater challenge for the movement of multiple mobile robots.

1. Simulation

The algorithm was first tested in Stage. Figure 47 shows a simulation where the leader is given the command to track a circle of radius 3 meters. The follower is not aware of the leader's intentions. It only receives pose information updates from the leader. The follower is tracking a goal state of 1 m directly behind the leader. The leader is shown in red, while the follower is in green. Notice that the follower moves in a slightly larger circle than the leader. This can be explained by the follower tracking a point 1 m directly behind the leader. Since the leader is constantly in a turn, the tracking point for the follower is cast out to a larger radius circle.

Other than the small tracking error in the circle, the follower performs remarkably well for such a simple algorithm. To test this method further, two different step inputs were given to the leader. In Fig. 48, a goal position of (2m, 3.5m, 0°) was given to the leader. Once the leader arrived at that location, a second pose of (5m, 1.5m, -90°) was given. While the leader was moving to these goal locations, it continuously sent its current pose to the follower.

Again, the follower performed very well. One thing to note, however, is that the follower tends to amplify the path of the leader. For the first step input, the leader moved within a single meter in the x direction. The follower required a slightly wider path. This is actually the same phenomenon observed in the first experiment, where the follower moved in a larger circle than the leader.

2. Experiment

Once the algorithm proved successful in the simulator, the experiment was performed in the testbed. The leader was given a circle to track. The initial poses of the two robots were just outside the circle, with the follower approximately 0.5 meters behind the leader. For spatial reasons, the tracking distance of the follower was decreased to 0.5 me-

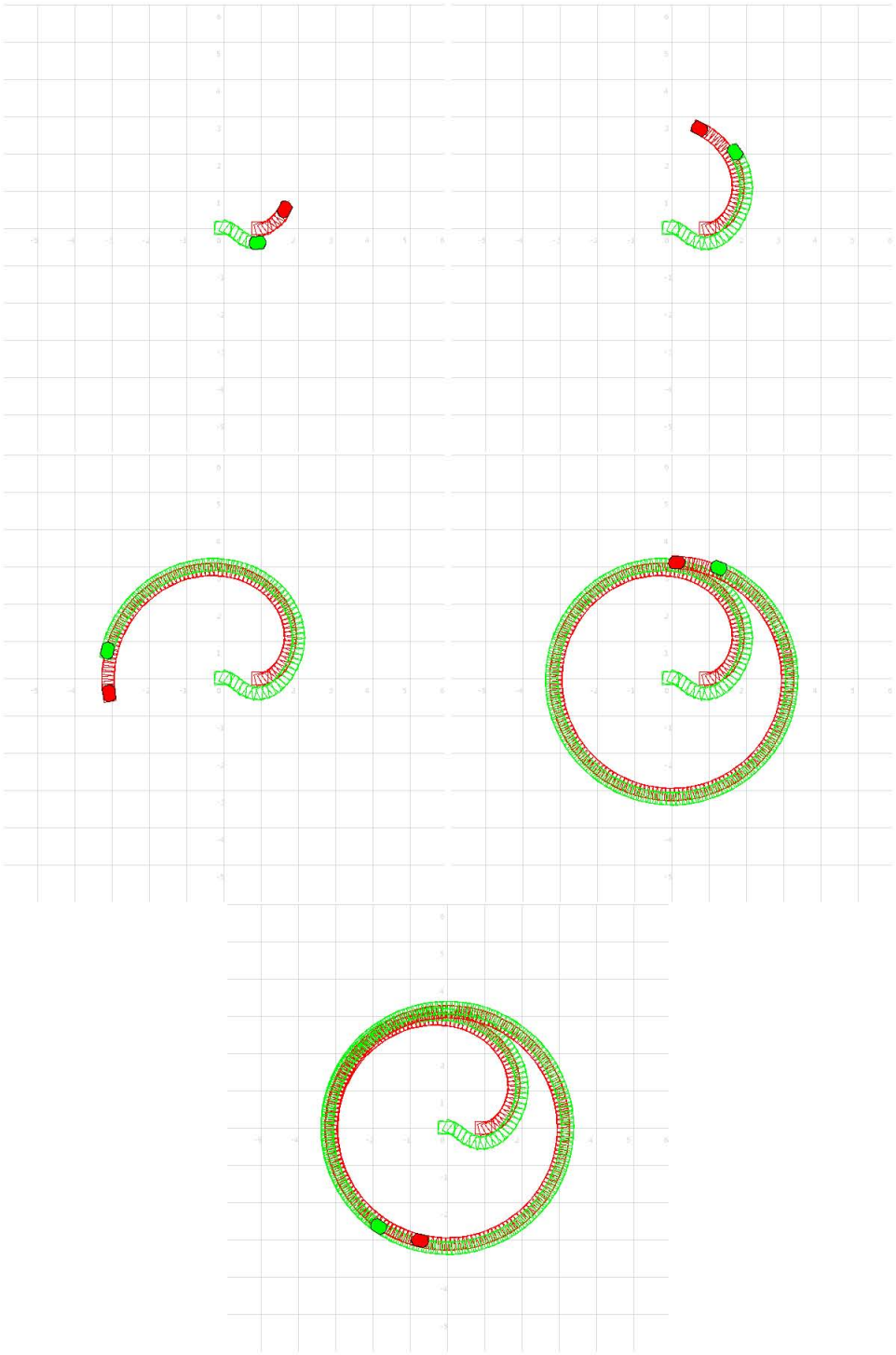

FIGURE 47 – Simulation of a leader-follow algorithm, with the leader given the objective to track a circle. The leader is shown in red and the follower in green. The initial position of the robots are near the origin as shown, with the follower 1 meter behind the leader.

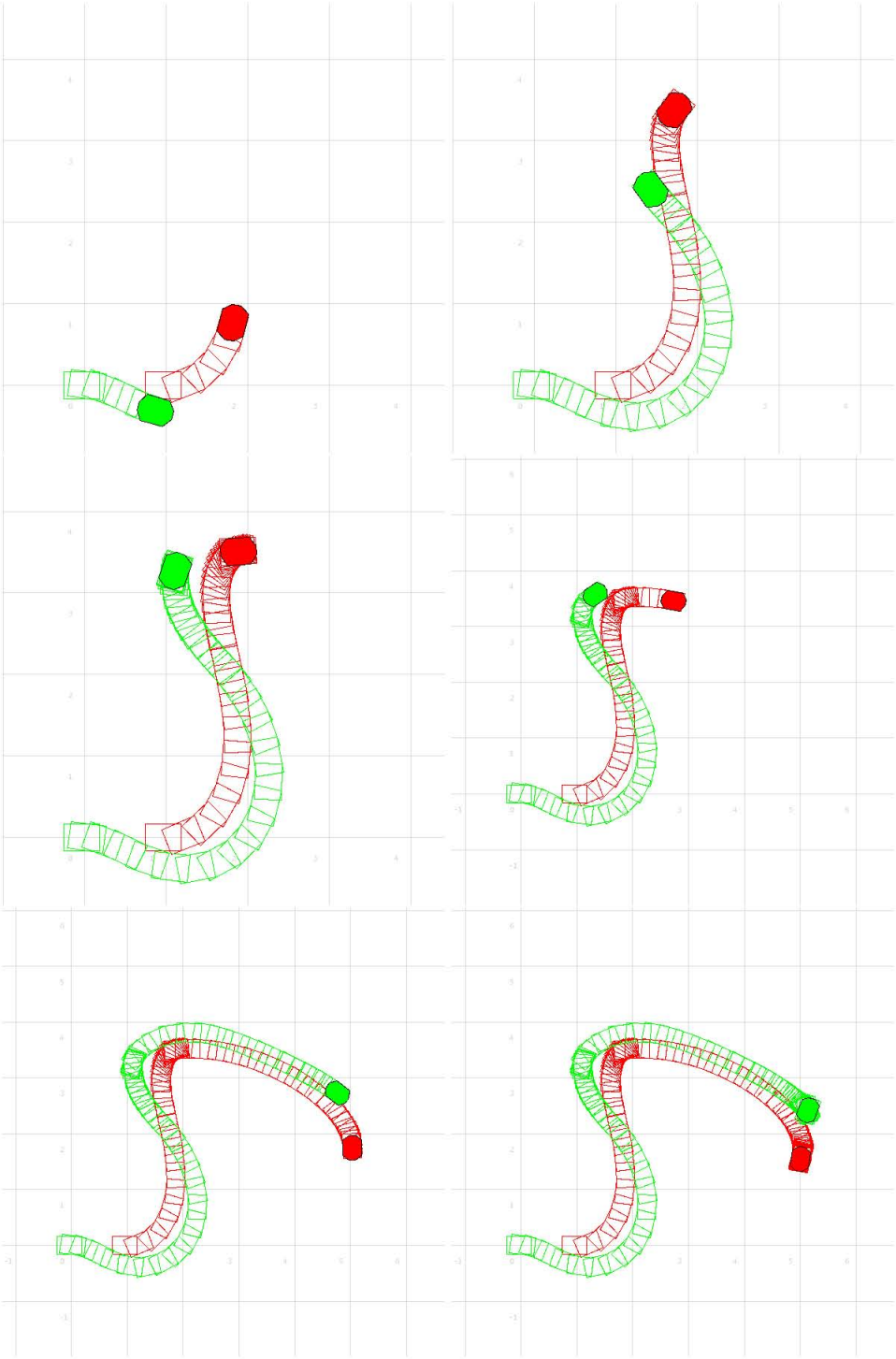

FIGURE 48 – Simulation of a leader-follow algorithm, with the leader given two separate movement commands. The first command was for the leader to go to the pose (2, 3.5, 0°). Once that was reached, a second command pose was given as (5, 1.5 -90°). The leader is shown in red and the follower in green.

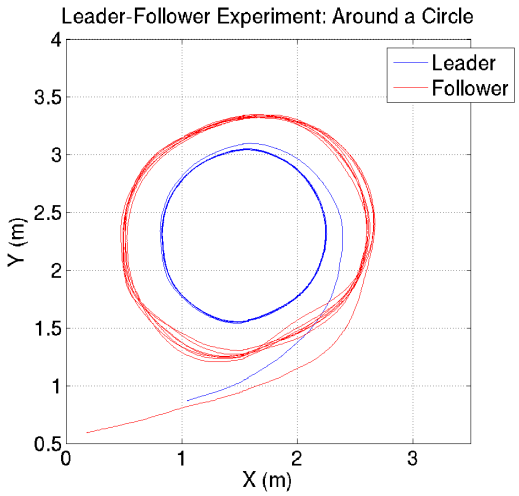

FIGURE 49 – Leader-Follow experiment with ER1 robots. The robot was given a circle to track. The initial location of the follower was approximately 0.5 m behind the leader.

ters. When a 1 meter following distance was used, the follower robot did not have enough room in the testbed.

Figure 49 shows the results of the experiment performed in the testbed. Approximately three minutes of experimental data is shown in the figure. The ER1 robots performed well. Again, the follower moves in a larger circle than the leader. This is more evident in the hardware experiment because a smaller circle was given to the leader. As a result, the follower goal state is cast farther away from the leader's circle. As the radius of the leader circle approaches infinity, the difference between the leader and follower paths approaches zero.

The leader moved in a very consistent circular path in this experiment. Figure 49 shows that the leader repeatedly drove within millimeters of the path. The follower robot was not as consistent as the leader. The red path appears to be a slightly noisier version of the leader's. This behavior was not apparent in the simulator. Unlike the simulator, some level of noise is always present in the hardware system. Noise causes slight variations in the control loop of the leader. The leader's controller does a good job of tracking the circle by making small corrections in angular velocity. These small changes in θ invoke a larger

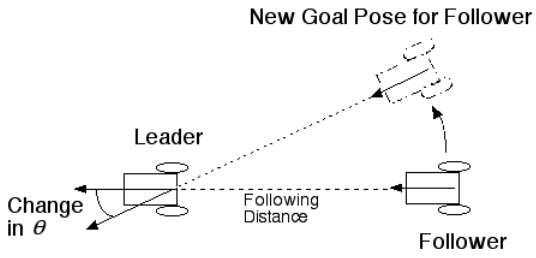

FIGURE 50 – Diagram of result of noise in the system. When the leader makes a small change in θ, the goal for the follower makes a larger change. This is the reason that there is more variance in the follower's path.

change in the goal position of the follower. Recall that the follower is tracking the position 0.5 meters directly behind the leader. Figure 50 shows this effect. As the leader rotates slightly, the goal pose of the follower shifts a considerable amount.

The robots depicted here are differential drive. They are constrained to move in the local x direction only. The new goal pose is shifted in the local y direction of the follower. In order to track the new goal, the follower is forced to rotate toward the shifted position. So, the unnoticeable change in the leader's θ results in a large change in the follower's θ. The leader continues to make these small changes in orientation as it tracks the circle. These small changes are magnified in the follower's path.

This affect is more noticeable when a second follower is added to the experiment. The follower algorithm is designed in a way that allows it to be easily extended. A second instance of the follower program is simply run on a new robot. The new robot tries to track the first follower. In this way, the new robot is unaware of the first leader and vice versa. This puts less burden on the leader robot. Now, the first follower simply communicates its own location to the new follower. Figure 51 shows the results of the experiment. The experiment is exactly the same as the previous, except that a new robot is added to the end of the line formation. The first two robots behave the same as before; moving in a nice formation. The third robot has more difficulty following for two reasons. First, the noise mentioned earlier propagates through the formation, growing as it moves from leader to

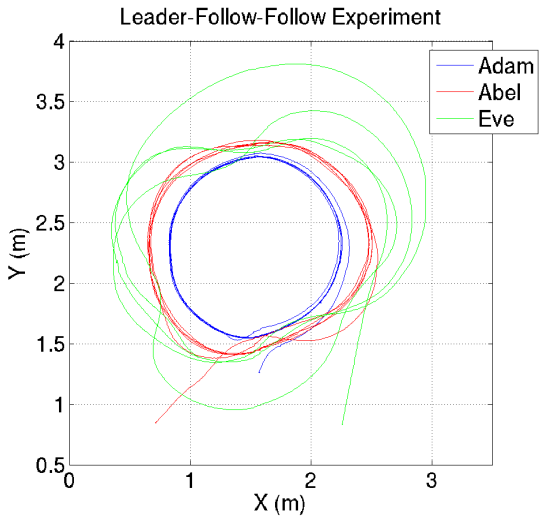

FIGURE 51 – Leader-Follow-Follow experiment with ER1 robots. The leader robot, Adam, shown in red, was given a circle to track. The first follower, Abel, shown in green tried to track Adam. The second follower, Eve, shown in green, tries to follow Abel. The last robot is the tractor-trailer robot design seen earlier.

followers. By the time the last robot receives its goal information, the data is quite noisy, forcing the robot to adjust to a continuously changing radius.

The second reason for the last robot's performance is its configuration. This robot is built with a hinge in its center. The hinge emulates a tractor-trailer. Unlike the other two ER1 robots, this robot is unable to rotate in place. The trailer reduces the degree of mobility by imposing an extra constraint on the robot's kinematics. So, not only does the third robot have a noisier path to follow, it is also constrained more than the other two robots. This explains the second follower's path, shown in green in Fig. 51.

CHAPTER VIII
EXTENSION TO 3D: A TESTBED DESIGN FOR SMALL-SCALE AERIAL VEHICLES

A. Need for a 3D Testbed

Until recently, three dimensional testbeds have been built [5] strictly for outdoor use with large aerial vehicles. They employ expensive GPS and inertial sensors to measure the position of the robots. With the recent availability of small, affordable, aerial vehicles, three dimensional testbeds are being built for indoor use. MIT recently purchased such a testbed from a company called Vicon [6]. Their new testbed uses high-speed, high-resolution, infrared cameras to track the position of multiple aerial vehicles. A similar design is presented here, which focuses on using small aerial vehicles in an indoor setting. It is will also use a vision system to track the pose of multiple robots efficiently.

B. Analysis & Design

1. From 2D to 3D

In the setup of a typical two dimensional testbed, cameras are mounted overhead and face downward to view the surface. It is customary that the viewing areas of the cameras overlap in order to ease calibration. During calibration, pixels are mapped to a new reference frame usually expressed in distances such as meters. Most of these systems make the assumption that the light rays entering the camera lens are perpendicular to the floor and parallel to each other. A camera located at infinity would receive such light rays;

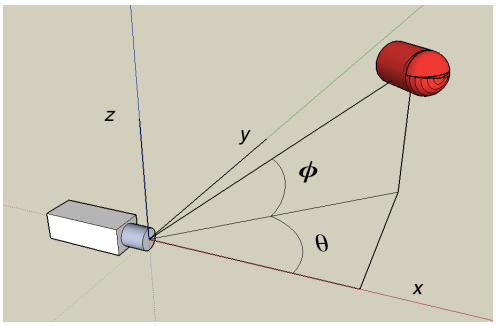

FIGURE 52 – Spherical coordinate system of the camera, with the camera at the origin and an aerial vehicle shown in the first octant. A light ray extends from the center of the vehicle to the camera lens. This light ray can be described by its spherical coordinate angles θ and ϕ.

however, the testbed cameras are obviously not located at infinity. Even so, this is usually a safe simplification; given that the cameras are mounted a fair distance from the surface of the testbed, and therefore a good distance from the robots themselves. An aerial vehicle will constantly vary its distance from an overhead camera. The closer the vehicle to the camera, the less valid the parallel light ray assumption.

A more accurate model of the incident light rays to the camera involves a change to the spherical coordinate system. Here, in Fig. 52, the camera represents the origin of the system and the light rays approach the lens at varying angles. Each light ray can be modeled with two angles. One, typically denoted θ, is the angle within the X-Y plane from the positive x-axis. The second, usually ϕ, is the angle from the X-Y plane.

Now, a camera is a two-dimensional sensor. Thus, in order to measure three dimensions, it is necessary for a robot to be covered by more than one camera at any given time. The difficulty lies in placing the cameras in such a way as to cover each point in the testbed space at least twice. The natural tendency to extending a two-dimensional testbed to three dimensions is to simply place cameras on a wall that look across the testbed. The overhead cameras could measure the x and y variables, while the wall cameras would measure the z position of the vehicle. This approach is limited by another factor of the camera: *field of view.* With the two-dimensional testbed, the *parallel ray* assumption ignored the field of view of the cameras. Having done away with this assumption, Figs. 53 and 54 show that the

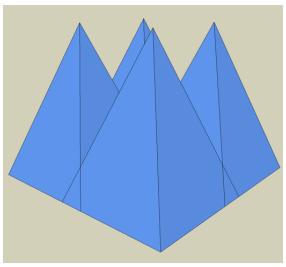

FIGURE 53 – Camera coverage of four cameras suspended from the ceiling. Each camera is located at the point of each "pyramid". The pyramids overlap creating this shape which shows the viewable space of the cameras.

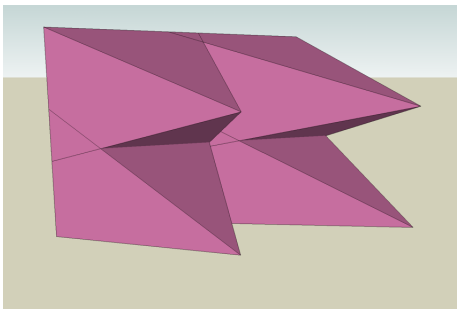

FIGURE 54 – Camera coverage of four cameras mounted on a wall. These would be used to capture the z value, or altitude, of a robot. Each camera is located at the tip of the "pyramid" shape which shows the camera's viewable space. Here the viewable spaces overlap slightly.

field of view plays a much larger role in the viewable space of the camera. In Figs. 53 and 54, four cameras are shown, each with a "pyramid" that represents the camera's viewable space. The camera is at the top of the pyramid. Each camera's pyramid overlaps with its neighbor's. Any space not contained by these shapes is not seen by the cameras. With the overhead cameras and the new side wall cameras, there are large, triangular prism-shaped areas that are unseen. If the aerial vehicle climbs too high, the overhead cameras will not be able to see the robot at all. The same is true as the vehicle gets closer to the side wall cameras.

A simple solution would be to add more cameras to cover the vacant area or to move the cameras outside of the testbed, so that there is a larger viewable space. However, this solution does not remedy the problem of the field of view. It also increases the cost of the system, both with hardware and processing power to support the hardware. There is a

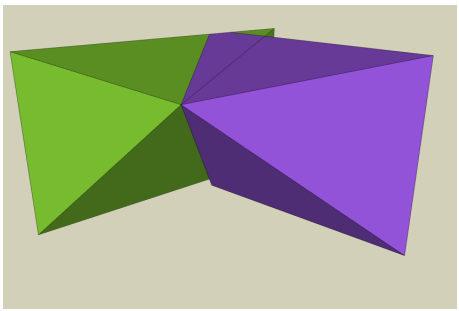

FIGURE 55 – This figure shows two cameras mounted very close to each other in an upper corner of the proposed testbed. They are angled downward slightly at the same ϕ angle but have different horizontal (or θ) angles. The viewable spaces overlap slightly. For clarity, the left camera's viewable space pyramid is a different color than the right camera's pyramid.

simple solution that consolidates the field of view gaps for a continuous region of coverage.

Instead of placing cameras along primary axes, the cameras could be positioned in the corners of the testbed. The cameras could then be oriented to account for the field of views and cover as much continuous area as possible. This is best explained with the help of another figure. In Fig. 55, it is easy to see that two cameras placed in close proximity, but at different orientation angles, cover a large majority of the testbed. Notice also that the region near the ceiling, where an aerial vehicle is most likely to fly, does not have a gap as before. Now, in order to capture an accurate three dimensional pose of the vehicle, it is necessary that at least two cameras can "see" the robot. Seeing little overlap between these two angled cameras, more cameras are obviously needed. That being so, two "mirrored" cameras are added in the adjacent corner in Fig. 56. Now, a large part of the volume can be seen by at least two cameras - enough to get the information needed.

2. Calculation of the Pose

When the vehicle of interest is seen by at least two cameras, the pose variables can be calculated using simple geometry. The three dimensional problem can be analyzed in two dimensions for each orthogonal angle θ and ϕ. In Fig. 57, an overhead view of the testbed with an object of interest spotted by two arbitrary cameras is shown. The center of

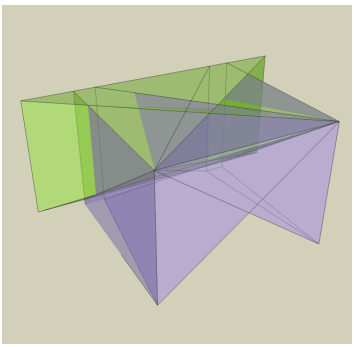

FIGURE 56 – The same as Fig. 55, except two more cameras are added to the adjacent corner. The pyramids are shown as semi-transparent to more easily see the total coverage. The entire upper section of the testbed is covered by the cameras.

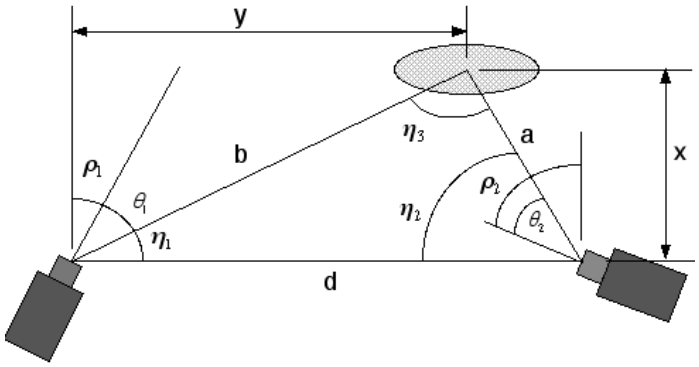

FIGURE 57 – An overhead view of two cameras mounted in adjacent corners a distance d from each other, at arbitrary angles ρ_1 and ρ_2. The ellipse shown is an aerial vehicle. The cameras are used to measure θ_1 and θ_2 and triangulate the vehicle's x and y coordinates.

mass of the object will correspond to a θ angle and a ϕ angle for each camera. Taking into account the orientation of each camera, ρ_1 and ρ_2, the angle between the intersecting light rays, η_3, can be calculated from (24), (25) and (26).

$$\eta_1 = 90^\circ - \theta_1 - \rho_1 \tag{24}$$

$$\eta_2 = 90^\circ + \theta_2 - \rho_2 \tag{25}$$

$$\eta_3 = -\eta_2 - \eta_1 + 180^\circ \tag{26}$$

The distance between the cameras, d, is known. So, using the law of sines, with the definition of the sine ratio, the distance x is found. Equation (27) allows for the other two legs of

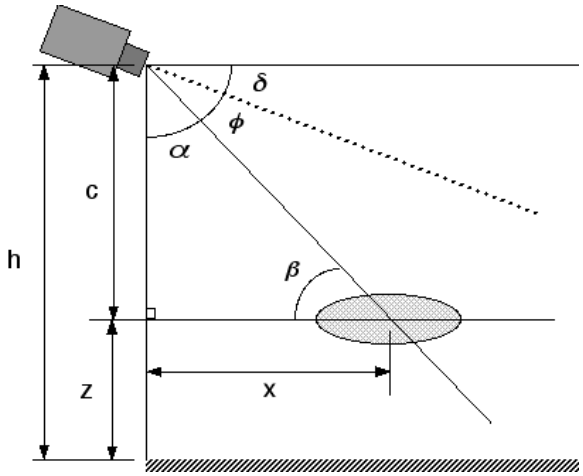

FIGURE 58 – The cameras from Fig. 57 seen from the side. They are both at the same position and downward orientation δ. The ellipse is the aerial vehicle. The cameras both measure the same angle ϕ, which is then used with the recently found value of x to calculate z, the altitude of the vehicle.

the triangle, a and b, to be calculated.

$$\frac{sin\ \eta_1}{a} = \frac{sin\ \eta_2}{b} = \frac{sin\ \eta_3}{d} \tag{27}$$

Finally, the desired x variable is calculated using the simple definition of the sine ratio.

$$x = b\ sin\ \eta_1 = a\ sin\ \eta_2 \tag{28}$$

The y component can be calculated just as easily with the cosine ratio.

$$y = b\ cos\ \eta_1 = a\ cos\ \eta_2 \tag{29}$$

In order to find the altitude, z, the side view is examined. Using the information found previously, x, and the angle ϕ, z can be calculated using the tangent ratio. Fig. 58 shows the side view of the cameras along with the object of interest. The angle from the horizontal to the camera's central axis is denoted as δ. The ray corresponding to the object of interest's center of mass is measured by the camera as ϕ. From the opposite interior angles of parallel lines theorem, β can be defined in (30) as

$$\beta = \mid \phi + \delta \mid \tag{30}$$

Once β is found, the altitude can be calculated with (31).

$$z = h - x \, tan \, \beta \tag{31}$$

Therefore, all of the position variables, x, y, and z, can be calculated using the view of two cameras. This allows an aerial vehicle to move through a large continuous space with a known, three-dimensional position.

A three dimensional, indoor testbed can be used to perform many types of robotics experiments. The size and price tag of aerial vehicles continues to shrink, allowing researchers more access to this type of craft for research. In order to develop new systems and perform reliable experiments, an instrument is needed to accurately measure the movement of aerial vehicles. The vision-based testbed presented here will be able to meet those needs. Aircraft can be tracked easily using the simple techniques that this testbed relies upon.

CHAPTER IX
CONCLUSIONS & FUTURE WORKS

A. Conclusions

This testbed is an enabling technology. It is a tool that can be used for advanced research in mobile robotics. The functions are two-fold. Control algorithms can be developed and then verified with a physical system. Here, a design has been presented for an accurate, real-time multi-robot testbed. Two separate designs for tracking vehicles were demonstrated. The implementation details were presented along with problems met and overcome.

Several difficulties were encountered with the cameras themselves. Trying to match the color output of four different cameras proved to be a challenge. The resulting outputs are close, but certainly not the same. This affected the thresholding of colors. A hardware configuration was found to separate the color levels for easier thresholding.

The camera lens distortion also became a major issue. It caused some of the algorithms to fail when the robots moved between different camera areas. This was eventually rectified by raising the cameras. There is still some distortion present, but not enough to interrupt the tracking.

Once the camera issues were discussed, the performance of the testbed was analyzed for alignment with the design goals of the apparatus. It was found to meet and surpass the original expectations and requirements of the testbed design.

As a demonstration of its function, several applications were explored covering control systems, modeling and autonomous coordinated movement. Several types of con-

trollers were developed to track circles and move to various poses. The testbed successfully provided position feedback for the mobile robots. It also proved to function much better than the robot's internal odometers as a source of reliable feedback.

Using robust identification algorithms, models for an individual robot were developed. The models presented closely matched the experimental data. The testbed also served well using multiple robots to test coordinated movement controllers in the leader-follow experiments. A method of decentralized control was investigated and verified using actual robot hardware. The algorithm was then extended to a leader-follow-follow formation with the ER1 robots.

Finally, a design was presented to extend the current testbed to three dimensions. This would allow the tracking of not only ground vehicles, but small, indoor, aerial vehicles as well. The experiments presented here could then be applied in three dimensions, allowing for much more advanced algorithms to be developed and verified.

B. Future Works

Now that the testbed is functional, many types of mobile robot research can be studied. With three robots available, coordinated formation movements can be investigated. Since the robots are heterogeneous in structure, control theories can be tested and verified quickly with different models. The simple controllers presented here can be replaced with much higher level controllers. Optimum and robust controllers can be developed to overcome the problems with the simple PD and PI designs of this thesis.

The leader-follow algorithm can be further extended or enhanced to provide better coordination. Instead of a robot simply tracking an offset distance behind a leader, maybe the follower could use the leader's pose information as "*waypoints*". In this way, the follower would actually try to follow the exact path of the leader. This could minimize the noise effect that was discussed previously, especially as the number of robots increases.

As far as the 3D testbed is concerned, only a theoretical design is presented here.

The next step is to actually construct it. The operation of this tool will be the ultimate test for the validity of the ideas presented here. One issue not addressed in this paper is the tracking of multiple vehicles in three dimensions. Multi-agent research is already popular for ground vehicles and becoming more popular for aerial vehicles as they become more available. The testbed will need the ability to track multiple aircraft.

The proposed 3D testbed will be able to track many vehicles, except in the case of vehicle occlusion. If one vehicle totally or partially blocks another vehicle from being seen by two cameras, the position of the occluded vehicle cannot be measured accurately, if at all. By adding cameras from other points of view and extending the principles presented here, this situation might be avoided in part. However, this does not truly solve the underlying problem.

The applications of this testbed are bound only by the imagination. Many new techniques can be developed with this flexible platform. They can then be verified using physical mobile robots. This ability is an invaluable asset to the advancement of robotics research. Will a robot eventually assist the elderly, or protect soldiers at the next war front? Will a machine search for precious resources on nearby planets? The answer to all of these is yes. The real question is when, but the answer is soon.

APPENDIX I

PROGRAM 1: Single Robot Tracking

```
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$
//$$              Single Robot Testbed Program
//$$                by Travis Riggs
//$$                 8/10/2006
//$$
//$$     This program grabs a video frame from 4 cameras through a Video
//$$     For Linux frame grabber. It uses some library functions
//$$     to communicate with the camera.
//$$
//$$     It will combine 4 frames together based hard coded values
//$$     from previous calibration. Then, it will calculate the
//$$     pose variables (x,y,theta) of the robot. It will then send
//$$     the data to the robot wirelessly.
//$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

#include "grabber.h"
#include "PracticalSocket.h"       // For UDPSocket Communications
#include <cstdlib>                 // For atoi()

#include <iostream>
#include <fstream>
#include <sstream>
#include <math.h>

//##########  Define New Image Types & Inline Functions  ###############
// Create a type to hold 2 pixels in packed YUV 422 packed format
typedef unsigned int YUYV32;

// Define functions for unpacking YUYV 422 format
#define Y0(n) (n & 0x000000ff)
#define CB(n) ((n >>  8) & 0x000000ff)
#define Y1(n) ((n >> 16) & 0x000000ff)
#define CR(n) ((n >> 24) & 0x000000ff)

// Define function for packing a pixel into YUYV 422 format
#define YUYV(y0,u,y1,v) ((v<<24)+(y1<<16)+(u<<8)+(y0))

// Create a type to hold and RGB pixel
typedef unsigned int RGB32;

// Define functions for unpacking RGB32 pixel
#define RED(n)   ((n >> 16) & 0x000000FF)
#define GREEN(n) ((n >> 8) & 0x000000ff)
#define BLUE(n)  ((n) & 0x000000ff)

// Define function for packing a RGB32 pixel
#define RGB(r,g,b) ((0<<24) + (r<<16) + (g <<8) + (b))

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@  Set Resolution Here  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@  Only one of these can be uncommented at a time. The     @@@@@@
//@@@@  desired resolution also needs to be set in Grabber.cpp  @@@@@@

//#####################################
// Full NTSC resolution
#define HIGHRES 1
int Height = NTSC_HEIGHT;
int Width  = NTSC_WIDTH;

#define CropLeft 20
#define CropRight 8
#define Pad 40

// Dimensions of cropped images
int NewHeight = NTSC_HEIGHT;
```

```
int NewWidth = NTSC_WIDTH - CropLeft - CropRight;

YUYV32 Frame0[NTSC_HEIGHT*NTSC_WIDTH/2];

// Dimensions of all 4 frames combined
int CombinedHeight = 960 + 2*Pad;
int CombinedWidth  = 1280 - 2*CropLeft - 2*CropRight + 2*Pad;
//#### End Full NTSC resolution #######

// //########################################
// // 1/4 NTSC Resolution
// #define HIGHRES 0
// int Height = NTSC_HEIGHT/2;
// int Width  = NTSC_WIDTH/2;
//
// //   !!!!! Need to be EVEN numbers  !!!!!!!
// #define CropLeft 10
// #define CropRight 4
// #define Pad 60
//
// // Dimensions of cropped images
// int NewHeight = NTSC_HEIGHT/2;
// int NewWidth  = NTSC_WIDTH/2 - CropLeft - CropRight;
//
// YUYV32 Frame0[NTSC_HEIGHT*NTSC_WIDTH/8];
//
// // Dimensions of all 4 frames combined
// int CombinedHeight = 480 + 2*Pad;
// int CombinedWidth  = 640 - 2*CropLeft - 2*CropRight + 2*Pad;
// //#### End 1/4 NTSC Resolution ###########

int main(int argc, char *argv[])
{

    //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    //@@@@@@@@@@ Variables   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

    int Debug = 1;   // Debug = 1, then only perform one iteration and print debug info

    int w, h;                  // Actual width and height of captured frame
    int i, j;                  // Counter variables
    int f, g;
    int color;                 // Another Counter
    int iter;
    int Row;
    int CaptureIterations;     // Number of measurements to take (command line argument)
    int FoundRobot = 0;        // Flag to indicate program found the robot

    char *name;                // Name to store picture files as ppm images
    char *Camera;              // Store device name of camera

    int R, G, B;               // Actual pixel values from frame in RGB space
    int Y, Cb, Cr;             // Calculated pixel values in YCbCr (YUV) space

    YUYV32  pixels;            // Structure to hold packed YUYV 422 pixels (2 pixels)
    RGB32   rgbpixel;

    // Array to store combined frames for searching
    YUYV32 Combined[(CombinedHeight)*(CombinedWidth)/2];

    // Array to store debugging image
    RGB32 ThreshPic[(CombinedHeight)*(CombinedWidth)];

    //========= Center of Mass Calculation Variables for each Color
    double RedCenterX = 0;
    double RedCenterY = 0;
    double RedMass = 0;

    int MinRedMass = 40;
    if(HIGHRES == 1)
    {
      MinRedMass = 300;
    }

    double BlueCenterX = 0;
    double BlueCenterY = 0;
    double BlueMass = 0;

    //========= Pose Variables of the Robot
    double RobotX;             // X position of robot (pixels)
    double RobotY;             // Y position of robot (pixels)
    double RobotTheta;         // Theta of Robot (degrees)

    //========= Conversion Variables
    // Calibration Factors to convert pixels to meters
    double YPixelsPerMeter = 128.62487; // Pixels per meter in the Y direction
    double XPixelsPerMeter = 133.98229; // Pixels per meter in the X direction
    if(HIGHRES == 1)
    {
      YPixelsPerMeter = 2*YPixelsPerMeter;
      XPixelsPerMeter = 2*XPixelsPerMeter;
```

```
}

double RobotXMeters;     // X Position of the robot in meters
double RobotYMeters;     // Y Position of the robot in meters

// Number of pixels to skip when searching for red circles
int SkipHoriz = 48; // This number needs to be a multiple of 4...actually twice the # to skip
int SkipVert  = 24; // This is the actual number to skip in the vertical direction
if(HIGHRES == 1)
{
  SkipHoriz = 2*SkipHoriz;
  SkipVert  = 2*SkipVert;
}

// Number of pixels for search box around a discovered Red pixel
//   !!!!! Need to be EVEN numbers  !!!!!!!
int SearchBox = 60;      // 16 will create a 2*16+1x2*16+1 or 33x33 search box
if(HIGHRES == 1)
{
  SearchBox = 2*SearchBox;
}

//========= Image Loop Variables
//   !!!!! Need to be EVEN numbers  !!!!!!!
int StartOffsetX = Pad + 2;    // Loop variables for searching combined frames.
int StartOffsetY = Pad + 2;    //  We don't want to waste time searching the black
int EndOffsetX = 500;      //  border (padding) of the combined image.
int EndOffsetY = 614;      //  These were found empirically.
if(HIGHRES == 1)
{
  EndOffsetX = 2*EndOffsetX;
  EndOffsetY = 2*EndOffsetY;
}

// Dimensions from Calibration to stitch frames together:
// Initialize Calibration Variables. When I first calibrated these values, I did it
//  at the floor level, but the hats sit about 14" above the floor. Due to the cone
//  angle, the floor level calibrations were cropping pieces of the hat off when the
//  robot would move from one camera to another on the floor. Here I'm using all the
//  true pixels (I cropped out the artifacts) I can get to ensure an accurate measure-
// -ment of the robots pose. Be careful if you change these, because each cameras uses
//  these values differently in the "for" loops below.

// X and Y point of center of testbed (in the lower right corner of Cam 0)
int XCal0 = 209;    //240;
int YCal0 = 280;    //320 - CropLeft - CropRight - 4;
if(HIGHRES == 1)
{
  XCal0 = 418;     //480;
  YCal0 = 561;     //640 - CropLeft - CropRight - 8;
}

// X & Y point of center of testbed (in lower left corner of Cam 1)
int XCal1 = 197;     //240;
int YCal1 = 21;      //10;
if(HIGHRES == 1)
{
  XCal1 = 395;       //480;
  YCal1 = 42;        //20;
}

// X & Y point of center of testbed (in upper right corner of Cam 2)
int XCal2 = 29;      //0;
int YCal2 = 280;     //320 - CropLeft - CropRight - 2;
if(HIGHRES == 1)
{
  XCal2 = 58;        //0;
  YCal2 = 561;       //640 - CropLeft - CropRight - 4;
}

// X & Y point of center of testbed (in upper left corner of Cam 3)
int XCal3 = 30;      //2;
int YCal3 = 26;      //10;
if(HIGHRES == 1)
{
  XCal3 = 60;      //4;
  YCal3 = 52;      //20;
}

//========== Thresholding Variables
int YThreshTable[256];     // Y Lookup Table to do fast constant thresholding
int CbThreshTable[256];    // Cb Lookup Table to do fast constant thresholding
int CrThreshTable[256];    // Cr Lookup Table to do fast constant thresholding

int ThresholdResult;    // Holds result of bitwise AND operations for thresholding

int WhiteMask = 0x01;  // Masks to access color lookup result
int RedMask   = 0x02;
int BlueMask  = 0x04;
int GreenMask = 0x08;
int BlackMask = 0x10;
```

```
int YFudge  = 0;        // Added variance for Y threshold levels
int CbFudge = 0;        // Added variance for Cb threshold levels
int CrFudge = 0;        // Added variance for Cr threshold levels

// Range of Threshold levels for each color and channel
//  Note: These can be set here, but they are later overwritten by data
//  from the ColorStats.txt file.  If you would like to set them manually,
//  you will need to comment out the 'Parsing Stats file' section below.
int RedYThreshLevelLow, RedYThreshLevelHigh;
int RedCbThreshLevelLow, RedCbThreshLevelHigh;
int RedCrThreshLevelLow, RedCrThreshLevelHigh;

int GreenYThreshLevelLow, GreenYThreshLevelHigh;
int GreenCbThreshLevelLow, GreenCbThreshLevelHigh;
int GreenCrThreshLevelLow, GreenCrThreshLevelHigh;

int BlueYThreshLevelLow, BlueYThreshLevelHigh;
int BlueCbThreshLevelLow, BlueCbThreshLevelHigh;
int BlueCrThreshLevelLow, BlueCrThreshLevelHigh;

int BlackYThreshLevelLow, BlackYThreshLevelHigh;
int BlackCbThreshLevelLow, BlackCbThreshLevelHigh;
int BlackCrThreshLevelLow, BlackCrThreshLevelHigh;

int WhiteYThreshLevelLow, WhiteYThreshLevelHigh;
int WhiteCbThreshLevelLow, WhiteCbThreshLevelHigh;
int WhiteCrThreshLevelLow, WhiteCrThreshLevelHigh;

int LowThreshold[5][3];   // Arrays to store lower and upper
int HighThreshold[5][3];  //  threshold bounds

int MinDataArray[5][3];   // To store min's of each color & channel
int MaxDataArray[5][3];   // To store max's of each color & channel
float MeanDataArray[5][3];    // To store mean of each color & channel
float StdDataArray[5][3];     // To store the Stdev of each color & chan.

//======== Data Storage File Streams
std::ofstream Tracking;            // File to store Pose Information of the Robot
std::ofstream RGBfile;             // RGB picture file in PPM format
std::ofstream Grayfile;            // PGM grayscale image

//======== Instantiate Grabber objects to grab frames
Grabber Grab0;              // For camera 0
Grabber Grab1;              // For camera 1
Grabber Grab2;              // For camera 2
Grabber Grab3;              // For camera 3

//========= Communication Variables
std::string servAddress = "192.168.1.111";       // Robot 1 (Adam) IP Address
char* echoString;               // Message to send to Robots
int echoStringLen;              // Length of string to echo  = strlen(echoString)
unsigned short echoServPort = 7000;     // Port to Communicate to Robot
const int ECHOMAX = 255;                // Longest string to echo
char message[200];

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@  End Variables  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
{
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//^^^^^^^          Parse Command Line Arguments     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

if (argc != 2) {        // Test for correct number of parameters
  cerr << "Usage: " << argv[0] << " NumberOfIterationsToCapture\n\n" << endl;
  cerr << " For example, to measure 37 locations with the\n";
  cerr << " camera system type:  ./simple 37\n\n";
  exit(1);
}

CaptureIterations = atoi(argv[1]);

//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$$   Parse Color Statistics File for Good Threshold Ranges  $$$$$$$$$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

// Open the file stream for reading
std::ifstream ColorDataFile;  // This file holds statistics for each color
std::string FileName = "../ColorStats.txt";
ColorDataFile.open(FileName.c_str());

if(!ColorDataFile)
{
  std::cout << "Error Opening Color File\n";
  return EXIT_FAILURE;
}

// Read data from the text file into
std::string Label;       // Labels in data file for humans to read
std::string Value;       // Actual number that we need

// Read each color (Red, Green, Blue, Black, White)
for(color = 0; color < 5; color++)
```

```
{
    // Read each channel (Y, Cb, Cr)
  for(i = 0; i < 3; i++)
  {
    // Read Minimum value
    std::getline(ColorDataFile, Label, '=');
    std::getline(ColorDataFile, Value, ',');
    MinDataArray[color][i] = atoi(Value.c_str());

    // Read Mean value
    std::getline(ColorDataFile, Label, '=');
    std::getline(ColorDataFile, Value, ',');
    MeanDataArray[color][i] = atof(Value.c_str());

    // Read Standard Deviation value
    std::getline(ColorDataFile, Label, '=');
    std::getline(ColorDataFile, Value, ',');
    StdDataArray[color][i] = atof(Value.c_str());

    // Read Maximum value
    std::getline(ColorDataFile, Label, '=');
    std::getline(ColorDataFile, Value, ',');
    MaxDataArray[color][i] = atoi(Value.c_str());

  }
}

// Close Color Data File
ColorDataFile.close();

//$$$$$$$$$$$$$$  End Parsing of Stats File  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

// Since we are using the Min and Max to set the range, I'm going to add a little
//  to each value just to capture a few more pixels that might not have been
//  present during the calibration.
int Fudge = 0;

for(color = 0; color < 5; color++)
{
  for(i = 0; i < 3; i++)
  {
    LowThreshold[color][i] = MinDataArray[color][i] - Fudge;
    HighThreshold[color][i] = MaxDataArray[color][i] + Fudge;

    // Check boundaries (has to be [0 255])
    if(MinDataArray[color][i] < 0)    LowThreshold[color][i] = 0;
    if(MaxDataArray[color][i] > 255)  HighThreshold[color][i] = 255;
  }
}

//  // Here, we are going to use the Mean and the Standard deviation to set the
//  //  threshold ranges automatically. We will include a range centered at the
//  //  mean that covers a multiple of the standard deviation from the mean.
//  double StdevMultiplier = 4.0;
//
//  for(color = 0; color < 5; color++)
//  {
//          // "i" is each channel (YCbCr)
//          for(i = 0; i < 3; i++)
//          {
//                  LowThreshold[color][i] = (int)(MeanDataArray[color][i] -
//                  StdevMultiplier*(double)StdDataArray[color][i]);
//                  HighThreshold[color][i] = (int)(MeanDataArray[color][i] +
//                  StdevMultiplier*(double)StdDataArray[color][i]);
//
//                  // Check bounds of thresholds
//                  if(LowThreshold[color][i] < 0)          LowThreshold[color][i] = 0;
//                  if(HighThreshold[color][i] > 255)       HighThreshold[color][i] = 255;
//          }
//  }

// Copy values from DataArrays into Threshold Levels. This may seem inefficient
//  but this step allows us to change the method we use to specify the
//  threshold levels, without a major change to the code.
RedYThreshLevelLow      =  LowThreshold[0][0];
RedCbThreshLevelLow     =  LowThreshold[0][1];
RedCrThreshLevelLow     =  LowThreshold[0][2];

RedYThreshLevelHigh     =  HighThreshold[0][0];
RedCbThreshLevelHigh    =  HighThreshold[0][1];
RedCrThreshLevelHigh    =  HighThreshold[0][2];

GreenYThreshLevelLow    =  LowThreshold[1][0];
GreenCbThreshLevelLow   =  LowThreshold[1][1];
GreenCrThreshLevelLow   =  LowThreshold[1][2];

GreenYThreshLevelHigh   =  HighThreshold[1][0];
GreenCbThreshLevelHigh  =  HighThreshold[1][1];
GreenCrThreshLevelHigh  =  HighThreshold[1][2];

BlueYThreshLevelLow     =  LowThreshold[2][0];
```

```
BlueCbThreshLevelLow    =  LowThreshold[2][1];
BlueCrThreshLevelLow    =  LowThreshold[2][2];

BlueYThreshLevelHigh    =  HighThreshold[2][0];
BlueCbThreshLevelHigh   =  HighThreshold[2][1];
BlueCrThreshLevelHigh   =  HighThreshold[2][2];

BlackYThreshLevelLow    =  LowThreshold[3][0];
BlackCbThreshLevelLow   =  LowThreshold[3][1];
BlackCrThreshLevelLow   =  LowThreshold[3][2];

BlackYThreshLevelHigh   =  HighThreshold[3][0];
BlackCbThreshLevelHigh  =  HighThreshold[3][1];
BlackCrThreshLevelHigh  =  HighThreshold[3][2];

WhiteYThreshLevelLow    =  LowThreshold[4][0];
WhiteCbThreshLevelLow   =  LowThreshold[4][1];
WhiteCrThreshLevelLow   =  LowThreshold[4][2];

WhiteYThreshLevelHigh   =  HighThreshold[4][0];
WhiteCbThreshLevelHigh  =  HighThreshold[4][1];
WhiteCrThreshLevelHigh  =  HighThreshold[4][2];

//%%%%%%%% Debug - Print the levels used for thresholding
if(Debug > 0)
{
  std::cout << "Red:\n";
  std::cout << "Y\t" << RedYThreshLevelLow << "\t" << RedYThreshLevelHigh << "\n";
  std::cout << "Cb\t" << RedCbThreshLevelLow << "\t" << RedCbThreshLevelHigh << "\n";
  std::cout << "Cr\t" << RedCrThreshLevelLow << "\t" << RedCrThreshLevelHigh << "\n";

  std::cout << "Green:\n";
  std::cout << "Y\t" << GreenYThreshLevelLow << "\t" << GreenYThreshLevelHigh << "\n";
  std::cout << "Cb\t" << GreenCbThreshLevelLow << "\t" << GreenCbThreshLevelHigh << "\n";
  std::cout << "Cr\t" << GreenCrThreshLevelLow << "\t" << GreenCrThreshLevelHigh << "\n";

  std::cout << "Blue:\n";
  std::cout << "Y\t" << BlueYThreshLevelLow << "\t" << BlueYThreshLevelHigh << "\n";
  std::cout << "Cb\t" << BlueCbThreshLevelLow << "\t" << BlueCbThreshLevelHigh << "\n";
  std::cout << "Cr\t" << BlueCrThreshLevelLow << "\t" << BlueCrThreshLevelHigh << "\n";

  std::cout << "Black:\n";
  std::cout << "Y\t" << BlackYThreshLevelLow << "\t" << BlackYThreshLevelHigh << "\n";
  std::cout << "Cb\t" << BlackCbThreshLevelLow << "\t" << BlackCbThreshLevelHigh << "\n";
  std::cout << "Cr\t" << BlackCrThreshLevelLow << "\t" << BlackCrThreshLevelHigh << "\n";

  std::cout << "White:\n";
  std::cout << "Y\t" << WhiteYThreshLevelLow << "\t" << WhiteYThreshLevelHigh << "\n";
  std::cout << "Cb\t" << WhiteCbThreshLevelLow << "\t" << WhiteCbThreshLevelHigh << "\n";
  std::cout << "Cr\t" << WhiteCrThreshLevelLow << "\t" << WhiteCrThreshLevelHigh << "\n";
}

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@   Create a Thresholding Array for up to 32 colors   @@@@@@@@@@@@@@@@@@@@
for(i = 0; i < 256; i++)
{
    YThreshTable[i]  = 0;        // Initialize arrays to zeros
    CbThreshTable[i] = 0;
    CrThreshTable[i] = 0;
}

// Place "1" in proper region for the Color Red for each channel (Y, Cb, Cr)
for(i = RedYThreshLevelLow; i <= RedYThreshLevelHigh; i++)
{
    YThreshTable[i] = YThreshTable[i] | RedMask;   // Y channel
}
for(i = RedCbThreshLevelLow; i <= RedCbThreshLevelHigh; i++)
{
    CbThreshTable[i] = CbThreshTable[i] | RedMask;
}
for(i = RedCrThreshLevelLow; i <= RedCrThreshLevelHigh; i++)
{
    CrThreshTable[i] = CrThreshTable[i] | RedMask;
}

// Place "1" in proper region for the Color Green for each channel (Y, Cb, Cr)
for(i = GreenYThreshLevelLow; i <= GreenYThreshLevelHigh; i++)
{
    YThreshTable[i] = YThreshTable[i] | GreenMask;   // Y channel
}
for(i = GreenCbThreshLevelLow; i <= GreenCbThreshLevelHigh; i++)
{
    CbThreshTable[i] = CbThreshTable[i] | GreenMask;
}
for(i = GreenCrThreshLevelLow; i <= GreenCrThreshLevelHigh; i++)
{
    CrThreshTable[i] = CrThreshTable[i] | GreenMask;
}

// Place "1" in proper region for the Color Blue for each channel (Y, Cb, Cr)
for(i = BlueYThreshLevelLow; i <= BlueYThreshLevelHigh; i++)
{
```

```
    YThreshTable[i] = YThreshTable[i] | BlueMask;    // Y channel
}
for(i = BlueCbThreshLevelLow; i <= BlueCbThreshLevelHigh; i++)
{
    CbThreshTable[i] = CbThreshTable[i] | BlueMask;
}
for(i = BlueCrThreshLevelLow; i <= BlueCrThreshLevelHigh; i++)
{
    CrThreshTable[i] = CrThreshTable[i] | BlueMask;
}

// Place "1" in proper region for the Color Black for each channel (Y, Cb, Cr)
for(i = BlackYThreshLevelLow; i <= BlackYThreshLevelHigh; i++)
{
    YThreshTable[i] = YThreshTable[i] | BlackMask;    // Y channel
}
for(i = BlackCbThreshLevelLow; i <= BlackCbThreshLevelHigh; i++)
{
    CbThreshTable[i] = CbThreshTable[i] | BlackMask;
}
for(i = BlackCrThreshLevelLow; i <= BlackCrThreshLevelHigh; i++)
{
    CrThreshTable[i] = CrThreshTable[i] | BlackMask;
}

// Place "1" in proper region for the Color White for each channel (Y, Cb, Cr)
for(i = WhiteYThreshLevelLow; i <= WhiteYThreshLevelHigh; i++)
{
    YThreshTable[i] = YThreshTable[i] | WhiteMask;    // Y channel
}
for(i = WhiteCbThreshLevelLow; i <= WhiteCbThreshLevelHigh; i++)
{
    CbThreshTable[i] = CbThreshTable[i] | WhiteMask;
}
for(i = WhiteCrThreshLevelLow; i <= WhiteCrThreshLevelHigh; i++)
{
    CrThreshTable[i] = CrThreshTable[i] | WhiteMask;
}
//@@@@@@@@  Finished Creating Threshold Table  @@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
}

//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$         Get Ready to Track Robot         $$$$$$$$$$$$$$$$$$$$$$

// Initialize RGB thresholded picture to White
if(Debug > 0)
{
  // Initialize to white
  for(i = 0; i < CombinedHeight*CombinedWidth; i++)
  {
    ThreshPic[i] = RGB(255, 255, 255);
  }
}


// Initialize some variables for Tracking the Robot
int HorizontalDiff0to2 = YCal0 - YCal2;
int VerticalDiff0to1 = XCal0 - XCal1;

// Setup the Frame Grabber and Cameras
// Open video device with YUV 422 format
Grab0.Init(CF_422, "/dev/video0", true);
Grab1.Init(CF_422, "/dev/video1", true);
Grab2.Init(CF_422, "/dev/video2", true);
Grab3.Init(CF_422, "/dev/video3", true);

// Set channel of multiplexor for input
Grab0.SetChannel("Composite0");
Grab1.SetChannel("Composite0");
Grab2.SetChannel("Composite0");
Grab3.SetChannel("Composite0");

// Video signal configure
Grab0.SetVideoSignal("ntsc");
Grab1.SetVideoSignal("ntsc");
Grab2.SetVideoSignal("ntsc");
Grab3.SetVideoSignal("ntsc");

Grab0.SetByteOrder(BYTE_ORDER_YUYV);
Grab1.SetByteOrder(BYTE_ORDER_YUYV);
Grab2.SetByteOrder(BYTE_ORDER_YUYV);
Grab3.SetByteOrder(BYTE_ORDER_YUYV);


if(Debug > 0)
{
  std::cout << "NewHeight = " << NewHeight << "\t";
  std::cout << "NewWidth = " << NewWidth << "\t";
  std::cout << "Combined Height = " << CombinedHeight << "\t";
  std::cout << "Combined Width = " << CombinedWidth << "\n";
}
```

```
// Open Text file to store pose information of the robot
Tracking.open("Data/PoseTracking.txt", std::ios::out);

// Start grabbing
Grab0.Start();
Grab1.Start();
Grab2.Start();
Grab3.Start();

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//%%%%%%%%%%%%   Begin Tracking Robot Loop   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for(iter = 0; iter < CaptureIterations; iter++)
{

  //#########################################################################
  //############   Crop Frames and Stitch Together   ########################
  //#########################################################################

  //@@@@ Upper Left Frame (Camera 0) @@@@@@@@@@@@@@@@@@@@@@@@*

  // Get frame from camera 0
  Grab0.Grab();

  // Quickly copy frame into array Frame0 for processing
  Grab0.CopyFrame((unsigned char *)&Frame0[0]);

  // Move through the image  while cropping the right and left
  //   "artifacts" created by the frame grabber
  for(i = 0; i < XCal0; i++)
  {
    Row = i*Width/2;

    for(j = 0; j < (YCal0 - CropLeft - CropRight)/2; j++)
    {

      // Copy the packed YUYV values into new, combined array
      Combined[(Pad+i)*CombinedWidth/2+(Pad/2)+j] = Frame0[Row+j+CropLeft/2];

    }
  }


  //@@@@ Upper Right Frame (Camera 1) @@@@@@@@@@@@@@@@@@@@@@@@

  // Get frame from camera 1
  Grab1.Grab();

  // Quickly copy frame into array Frame1 for processing
  Grab1.CopyFrame((unsigned char *)&Frame0[0]);

  // Move through the image while cropping the right and left
  //   "artifacts" created by the frame grabber
  for(i = 0; i < XCal1; i++)
  {
    Row = i*Width/2;

    for(j = 0; j < (NewWidth-YCal1)/2; j++)
    {

      // Repack the RGB values into new, cropped array
      Combined[(Pad+i+VerticalDiff0to1)*CombinedWidth/2+
               (Pad+CropRight-3+YCal0-YCal1)/2+j] = Frame0[Row+j+(YCal1+CropLeft)/2];

    }
  }


  //@@@@ Lower Left Frame (Camera 2) @@@@@@@@@@@@@@@@@@@@@@@@

  // Get frame from camera 2
  Grab2.Grab();

  // Quickly copy frame into array Frame2 for processing
  Grab2.CopyFrame((unsigned char *)&Frame0[0]);

  // Move through the image while cropping the right and left
  //   "artifacts" created by the frame grabber
  for(i = XCal2; i < NewHeight; i++)
  {
    Row = i*Width/2;

    for(j = 0; j < (YCal2 - CropLeft - CropRight)/2; j++)
    {

      // Repack the RGB values into new, cropped array
      Combined[(Pad+XCal0-XCal2+i)*CombinedWidth/2+
               (Pad+HorizontalDiff0to2)/2+j] = Frame0[Row+j+(CropLeft)/2];
```

```
  }
}

//@@@@ Lower Right Frame (Camera 3) @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

// Get frame from camera 3
Grab3.Grab();

// Quickly copy frame into array Frame3 for processing
Grab3.CopyFrame((unsigned char *)&Frame0[0]);

// Move through the image while cropping the right and left
//   "artifacts" created by the frame grabber
for(i = XCal3; i < NewHeight; i++)
{
  Row = i*Width/2;

  for(j = 0; j < (NewWidth-YCal3)/2; j++)
  {

    // Repack the RGB values into new, cropped array
    Combined[(Pad+XCal0-XCal3+i)*CombinedWidth/2+(Pad+HorizontalDiff0to2+
                 CropLeft+YCal0-YCal3)/2+j] = Frame0[Row+j+(YCal3+CropLeft)/2];

  }
}

//############  End Cropping & Stitching  ######################################
//##############################################################################


//####### Save the Combined Image as a Grayscale PGM
// Note: This should really be converted to a function, but I was
//  unsuccessful at accomplishing this because of scope issues.
if(Debug > 0)
{
  // Open file stream for write operations to save PGM file
  Grayfile.open("Frames/combined.pgm", std::ios::out);

  // Write Header for file
  Grayfile << "P2" << std::endl;
  Grayfile << CombinedWidth << " " << CombinedHeight << "\n";
  Grayfile << "255" << std::endl;

  // Copy Y pixels into file
  for(i = 0; i < CombinedHeight; i++)
  {
    for(j = 0; j < CombinedWidth; j = j + 2)
    {
      // Get 2 packed pixels from image
      pixels = Combined[i*CombinedWidth/2+j/2];

      // Extract first intensity value
      Y = Y0(pixels);
      // Write to file
      Grayfile << Y << " ";
      // Extract 2nd intenisty value
      Y = Y1(pixels);
      // Write to file
      Grayfile << Y << " ";
    }
  }

  // Close file
  Grayfile.close();
}

//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&&&&  Search for Robots  &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

// Initialize Tracking Variables
BlueCenterX = 0;
BlueCenterY = 0;
BlueMass    = 0;
RedCenterX  = 0;
RedCenterY  = 0;
RedMass     = 0;

// Clear search flag
FoundRobot = 0;

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@  Search for Red Rectangle in Combined Image  @@@@@@@@@@@@@

// Begin a sparse search of the testbed
for(i = StartOffsetX; i < EndOffsetX; i = i + SkipVert)
{
  for(j = StartOffsetY; j < EndOffsetY; j = j + SkipHoriz/2)
  {
    if(Debug > 4)
```

```
{
  std::cout << "i = " << i << " j = " << j << std::endl;
}

// Get 2 compressed Pixels from Framegrabber
pixels = Combined[i*CombinedWidth/2+j/2];

//%%%%%%% Need to include an if statement to test whether
//%%%%%%%  the program needs an even or an odd pixel. Then
//%%%%%%%  it should unpack the appropriate one.

// If the pixel requested is even
if( j % 2 == 0)
{
  // Extract even numbered Intensity pixel -> Y0
  Y = Y0(pixels);
}
//...of if the number is odd
else if( j % 2 == 1)
{
  // Extract odd numbered Intensity pixel -> Y1
  Y = Y1(pixels);
}

// Extract the color component of the pixels
Cb = CB(pixels);
Cr = CR(pixels);

// Threshold the image using lookup tables and bitwise AND statements
ThresholdResult = YThreshTable[Y] & CbThreshTable[Cb] & CrThreshTable[Cr];

if(ThresholdResult & RedMask)
{

  if(Debug > 1)
  {
    std::cout << "\nDetected Red at " << i << ", " << j << "\n";

  }

  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  //@@@@@@@@@@@@@@ Begin Search for Robot's Center @@@@@@@@@@@@@@@@@@@@
  // If we found a red pixel, then do a refined local search for
  //  the center of this robot
  for(f = -SearchBox; f <= SearchBox; f++)
  {
    for(g = -SearchBox; g <= SearchBox; g = g + 2)
    {

      //@!!!!!! We need to start on an EVEN pixel to insure accuracy
      //@!!!!!!  of center of mass calculation !!!!!!!!
      // If this is the first iteration....
      if(g == -SearchBox)
      {
        // If the starting Y coord is odd...
        if(-g % 2 == 1)
        {
          //scoot over 1 pixel to right to make it even
          g = -SearchBox + 1;
        }
      }

      // Get 2 compressed Pixels from Framegrabber
      pixels = Combined[(i+f)*CombinedWidth/2+(j+g)/2];

      // Extract first pixel from packed pixel
      Y  = Y0(pixels);
      Cb = CB(pixels);
      Cr = CR(pixels);

      // Threshold the image using lookup tables and bitwise AND statements
      ThresholdResult = YThreshTable[Y] & CbThreshTable[Cb] & CrThreshTable[Cr];

      if(ThresholdResult & RedMask)
      {

        // Keep track of Red mass
        // Sum all x coordinates
        RedCenterX = RedCenterX + (i + f);
        // Sum all y coordinates
        RedCenterY = RedCenterY + (j + g);
        RedMass++;                    // Track mass of Red pixels

        if(Debug > 10)
        {
          std::cout << "\nFound RED pix at ";
          std::cout << i+f << " " << j+g;
        }

        if(Debug > 0)
        {
          // Also store in Thresholded image as RED
```

```
      ThreshPic[(i+f)*CombinedWidth+(j+g)] = RGB(255, 0, 0);
    }

  }
  else if(ThresholdResult & BlueMask)
  {
    // Keep track of Blue mass
    // Sum all x coordinates
    BlueCenterX = BlueCenterX + (i + f);
    // Sum all y coordinates
    BlueCenterY = BlueCenterY + (j + g);
    BlueMass++;                       // Track mass of Blue pixels

    // Store thresholded result as pure YUYV Blue Pixels
    Combined[(i+f)*CombinedWidth/2+(j+g)/2] = YUYV(76, 255, 76, 85);

    if(Debug > 10)
    {
      std::cout << "\nFound BLUE pix at ";
      std::cout << i+f << " " << j+g;
    }

    if(Debug > 0)
    {
      // Also store in Thresholded image as BLUE
      ThreshPic[(i+f)*CombinedWidth+(j+g)] = RGB(0, 0, 255);
    }
  }
  else
  {

    if(Debug > 0)
    {
      // Pixel was unclassified, so set to Black
      ThreshPic[(i+f)*CombinedWidth+(j+g)] = RGB(0, 0, 0);
    }

  }

  // Extract second pixel from packed pixel
  Y  = Yl(pixels);
  Cb = CB(pixels);
  Cr = CR(pixels);

  // Threshold the image using lookup tables and bitwise AND statements
  ThresholdResult = YThreshTable[Y] & CbThreshTable[Cb] & CrThreshTable[Cr];

  if(ThresholdResult & RedMask)
  {

    // Keep track of Red mass
    // Sum all x coordinates
    RedCenterX = RedCenterX + (i + f);
    // Sum all y coordinates (remember to add 1 b/c it's 2nd pixel)
    RedCenterY = RedCenterY + (j + g + 1);
    RedMass++;                        // Track mass of Red pixels

    // Store thresholded result as pure YUYV Red Pixels
    Combined[(i+f)*CombinedWidth/2+(j+g)/2] = YUYV(76, 85, 76, 255);

    // For fastest execution
    if(Debug == 0)
    {
      continue;
    }

    if(Debug > 10)
    {
      std::cout << "\nFound RED pix at ";
      std::cout << i+f << " " << j+g+1;
    }

    if(Debug > 0)
    {
      // Also store in Thresholded image as RED
      ThreshPic[(i+f)*CombinedWidth+(j+g+1)] = RGB(255, 0, 0);
    }
  }
  else if(ThresholdResult & BlueMask)
  {
    // Keep track of Blue mass
    // Sum all x coordinates
    BlueCenterX = BlueCenterX + (i + f);
    // Sum all y coordinates (remember to add 1 b/c it's 2nd pixel)
    BlueCenterY = BlueCenterY + (j + g + 1);
    BlueMass++;                       // Track mass of Blue pixels

    // Store thresholded result as pure YUYV Blue Pixels
    Combined[(i+f)*CombinedWidth/2+(j+g)/2] = YUYV(76, 255, 76, 85);

    if(Debug > 10)
    {
```

```
          std::cout << "\nFound BLUE pix at ";
          std::cout << i+f << " " << j+g+1;
        }

        if(Debug > 0)
        {
          // Also store in Thresholded image as BLUE
          ThreshPic[(i+f)*CombinedWidth+(j+g+1)] = RGB(0, 0, 255);
        }
      }
      else
      {

        if(Debug > 0)
        {
          // Pixel was unclassified, so set to Black
          ThreshPic[(i+f)*CombinedWidth+(j+g+1)] = RGB(0, 0, 0);
        }

      }

    }
  } // End Detailed Search

  if(RedMass < MinRedMass)
  {
    // In case this search was started by some noise
    //  ...check for that using a "size filter"
    continue;
  }

  // Calculate Center of Mass for the Red Rectangle
  RedCenterX = RedCenterX/RedMass;
  RedCenterY = RedCenterY/RedMass;

  if(Debug > 0)
  {
    // Put a Green Dot at center of red mass in ThreshPic
    ThreshPic[(int)RedCenterX*CombinedWidth+(int)RedCenterY] = RGB(0, 255, 0);
  }

  // Calculate Center of Mass for the Blue Rectangle
  BlueCenterX = BlueCenterX/BlueMass;
  BlueCenterY = BlueCenterY/BlueMass;

  if(Debug > 0)
  {
    // Put a Green Dot at center of Blue mass in ThreshPic
    ThreshPic[(int)BlueCenterX*CombinedWidth+(int)BlueCenterY] = RGB(0, 255, 0);
  }

  // Calculate the Pose of the robot
  RobotX = (RedCenterX + BlueCenterX)/2;
  RobotY = (RedCenterY + BlueCenterY)/2;

  RobotXMeters = RobotX/XPixelsPerMeter;
  RobotYMeters = RobotY/YPixelsPerMeter;

  // The Theta calculation assumes that the Red block is worn on the front of
  // the robot and the blue block is worn on the back of the robot hat. This
  // makes the code easier to read because we don't have any weird offsets or
  // negations.  It is simply the arctangent of the line from the blue to red.
  RobotTheta = (180/M_PI)*atan2((RedCenterY - BlueCenterY),(RedCenterX - BlueCenterX));

  // Check proper bounds of Theta (-180, 180]
  if(RobotTheta > 180)  RobotTheta = RobotTheta - 360;
  if(RobotTheta < -180) RobotTheta = RobotTheta + 360;

  //====== Print robot's Pose Data to screen
  std::cout << "i: " << iter << " ";
  std::cout << "X = " << RobotX << " pixels = " << RobotXMeters << " m ";
  std::cout << "\tY = " << RobotY << " pixels = " << RobotYMeters << " m ";
  std::cout << "\tTheta = " << RobotTheta << std::endl;

  //====== Write Pose data to file in "PoseTracking.txt"
  Tracking << RobotX << "\t" << RobotXMeters << "\t";
  Tracking << RobotY << "\t" << RobotYMeters << "\t" << RobotTheta << "\n";

  //====== Send coordinates to Robot
  std::sprintf(message, "%f %f %f", RobotXMeters, RobotYMeters, M_PI*(RobotTheta)/180);

  echoString = message;
  echoStringLen = strlen(echoString);

  try {
    UDPSocket sock;

    // If the first iteration, send the "Go" signal (1) to the robot
    if(iter == 0)
    {
      echoString = "1";
```

```
          //******Debug
          std::cout << "Sending Go Flag...\n";
        }

        // Send the string to the robot
        sock.sendTo(echoString, echoStringLen, servAddress, echoServPort);

        // Destructor closes the socket

      } catch (SocketException &e) {
        cerr << e.what() << endl;
        exit(1);
      }

      // By the end of the detailed search, the robot should be found.
      //  So set a flag to stop the rest of the image search
      FoundRobot = 1;
      break;

      //@@@@@@@@@@@@@@  End Search for Robot's Center  @@@@@@@@@@@@@@@@@@@@@@
      //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

    }
    else
    {

      // For fastest execution
      if(Debug == 0)
      {
        continue;
      }

      // Store in Thresholded Picture as Black
      ThreshPic[i*CombinedWidth+j] = RGB(0, 0, 0);

    }

  } // Inner Sparse Search Loop

  // Have we already found the robot?
  if(FoundRobot == 1)
  {
    // Stop Looking and move to next set of frames
    break;
  }

}  // Outer Sparse Search Loop


//#####################################################################
//     Save the Combined Image as a Grayscale PGM
// Note: This should really be converted to a function, but I was
//  unsuccessful at accomplishing this because of scope issues.
if(Debug > 0)
{
  // Open file stream for write operations to save PGM file
  Grayfile.open("Frames/searched.pgm", std::ios::out);

  // Write Header for file
  Grayfile << "P2" << std::endl;
  Grayfile << CombinedWidth << " " << CombinedHeight << "\n";
  Grayfile << "255" << std::endl;

  // Copy Y pixels into file
  for(i = 0; i < CombinedHeight; i++)
  {
    for(j = 0; j < CombinedWidth; j = j + 2)
    {
      // Get 2 packed pixels from image
      pixels = Combined[i*CombinedWidth/2+j/2];

      // Extract first intensity value
      Y = Y0(pixels);
      // Write to file
      Grayfile << Y << " ";
      // Extract 2nd intenisty value
      Y = Y1(pixels);
      // Write to file
      Grayfile << Y << " ";
    }
  }

  // Close file
  Grayfile.close();
}

//#####################################################################
//     Save the Thresholded Image as an RGB PPM
// Note: This should really be converted to a function, but I was
//  unsuccessful at accomplishing this because of scope issues.
if(Debug > 0)
```

```
        {
          // Open file stream for write operations to save PGM file
          RGBfile.open("Frames/threshpic.ppm", std::ios::out);

          // Write Header for file
          RGBfile << "P3" << std::endl;
          RGBfile << CombinedWidth << " " << CombinedHeight << "\n";
          RGBfile << "255" << std::endl;

          // Copy Y pixels into file
          for(i = 0; i < CombinedHeight; i++)
          {
            for(j = 0; j < CombinedWidth; j++)
            {
              // Get 2 packed pixels from image
              rgbpixel = ThreshPic[i*CombinedWidth+j];

              // Extract channel values
              R = RED(rgbpixel);
              G = GREEN(rgbpixel);
              B = BLUE(rgbpixel);

              // Write to file
              RGBfile << R << " " << G << " " << B << " ";

            }
          }

          // Close file
          RGBfile.close();
        }

      }
      //%%%%%%%%%%%%%%%%   End Fast Robot Tracking Loop   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      // Stop Frame Grabber
      Grab0.Stop();
      Grab1.Stop();
      Grab2.Stop();
      Grab3.Stop();

      std::cout << "\n*********** All Done! ***********************\n\n";

      // Close Pose Tracking Text file
      Tracking.close();

      return 0;
}
```

APPENDIX II

PROGRAM 2: Multi-Robot Tracking

```
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$
//$$              Multi Robot Testbed Program
//$$                by Travis Riggs
//$$                  10/12/2006
//$$
//$$     This program grabs a video frame from 4 cameras through a Video
//$$     For Linux frame grabber. It uses some library functions
//$$     to communicate with the framegrabber.
//$$
//$$     It will examine 4 frames individually in search of robots.
//$$     Then, it will calculate the pose variables (x,y,theta) of each
//$$     robot. It will then send the data to the robots wirelessly.
//$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

//%%%%%%%%%%%%   Header Files   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#include <iostream>
#include <fstream>
#include <sstream>
#include <math.h>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <cstdlib>                  // For atoi()
#include <assert.h>

#include "grabber.h"
#include "PracticalSocket.h"        // For UDPSocket Communications


//##########  Define New Image Types & Inline Functions  ###############
// Create a type to hold 2 pixels in packed YUV 422 packed format
typedef unsigned int YUYV32;

// Define functions for unpacking YUYV 422 format
#define Y0(n) (n & 0x000000ff)
#define CB(n) ((n >>  8) & 0x000000ff)
#define Y1(n) ((n >> 16) & 0x000000ff)
#define CR(n) ((n >> 24) & 0x000000ff)

// Define function for packing a pixel into YUYV 422 format
#define YUYV(y0,u,y1,v) ((v<<24)+(y1<<16)+(u<<8)+(y0))

// Create a type to hold and RGB pixel
typedef unsigned int RGB32;

// Define functions for unpacking RGB32 pixel
#define RED(n)   ((n >> 16) & 0x000000FF)
#define GREEN(n) ((n >> 8) & 0x000000ff)
#define BLUE(n)  ((n) & 0x000000ff)

// Define function for packing a RGB32 pixel
#define RGB(r,g,b) ((0<<24) + (r<<16) + (g <<8) + (b))

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@  Set Resolution Here  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@  Only one of these can be uncommented at a time. The     @@@@@@
//@@@@  desired resolution also needs to be set in Grabber.cpp  @@@@@@

//#####################################
// Full NTSC resolution
#define HIGHRES 1
int Height = NTSC_HEIGHT;
int Width  = NTSC_WIDTH;

#define CropLeft 20
#define CropRight 8
#define Pad 20

// Dimensions of cropped images
int NewHeight = NTSC_HEIGHT;
```

```
int NewWidth = NTSC_WIDTH - CropLeft - CropRight;

YUYV32 Frame[NTSC_HEIGHT*NTSC_WIDTH/2];

// Dimensions of all 4 frames combined
int CombinedHeight = 960 + 2*Pad;
int CombinedWidth  = 1280 - 2*CropLeft - 2*CropRight + 2*Pad;

//Dimensions of single padded frames
int PaddedHeight = NTSC_HEIGHT + 2*Pad;
int PaddedWidth  = NTSC_WIDTH - CropLeft - CropRight + 2*Pad;
//####  End Full NTSC resolution  #######

// //#######################################
// // 1/4 NTSC Resolution
// #define HIGHRES 0
// int Height = NTSC_HEIGHT/2;
// int Width  = NTSC_WIDTH/2;
//
// //   !!!!! Need to be EVEN numbers  !!!!!!!
// #define CropLeft 10
// #define CropRight 4
// #define Pad 20
//
// // Dimensions of cropped images
// int NewHeight = NTSC_HEIGHT/2;
// int NewWidth  = NTSC_WIDTH/2 - CropLeft - CropRight;
//
// YUYV32 Frame[NTSC_HEIGHT*NTSC_WIDTH/8];
//
// // Dimensions of all 4 frames combined
// int CombinedHeight = 480 + 2*Pad;
// int CombinedWidth  = 640 - 2*CropLeft - 2*CropRight + 2*Pad;
// //####  End 1/4 NTSC Resolution  ###########


//###########################################
//######  Robot Constants  #################
// Maximum number of robots to search for
#define Robots 81
#define ExistingRobots 3

// !!!!! Need to divide evenly into 360  !!!!!!!
#define DegreeStep 1

//######   End Constants   #################
//###########################################

int main(int argc, char *argv[])
{

  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  //@@@@@@@@@@  Variables  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

    int Debug = 1;               // Debug > 0 print debug info
    int ErrorCorrection = 0;     // 1= Robot ID Error Correction ON (0= off)

    //====== Counter Variables
    int w, h;                    // Actual width and height of captured frame
    int i, j;                    // Counter variables
    int f, g;                    // Small loop counter variables
    int a, r;
    int Row;
    int color;                   // Another Counter
    int iter;
    int Camera;
    int CaptureIterations;       // Number of measurements to take (command line argument)
    int AlreadyFound = 0;        // Flag to indicate whether robot is already found or not

    char *name;                  // Name to store picture files as ppm images

    int R, G, B;                 // Actual pixel values from frame in RGB space
    int Y, Cb, Cr;               // Actual pixel values in YCbCr (YUV) space

    YUYV32  pixels;          // Structure to hold packed YUYV 422 pixels (2 pixels)
    RGB32   rgbpixel;

    // Instantiate Grabber objects to grab frames
    Grabber Grab0;       // For camera 0
    Grabber Grab1;       // For camera 1
    Grabber Grab2;       // For camera 2
    Grabber Grab3;       // For camera 3

    // Array to store combined frames for searching
    YUYV32 Padded0[(PaddedHeight)*(PaddedWidth)/2];
    YUYV32 Padded1[(PaddedHeight)*(PaddedWidth)/2];
    YUYV32 Padded2[(PaddedHeight)*(PaddedWidth)/2];
    YUYV32 Padded3[(PaddedHeight)*(PaddedWidth)/2];

    YUYV32 *FramePtr;    // Points to current camera frame
```

```
// Array to store debugging image
RGB32 ThreshPic[(PaddedHeight)*(PaddedWidth)];

//====== Pose Variables of the Robot
double RobotX[Robots];          // X position of robot (pixels)
double RobotY[Robots];          // Y position of robot (pixels)
double Theta[Robots];           // Theta of Robot (degrees)

//====== Center of Mass Calculation Variables for Red
double RedCenterX;
double RedCenterY;
double RedMass;

int MinRedSize = 70;     // Size filter for discarding red noise
if(HIGHRES == 1)
{
  MinRedSize = 200;
}

// Number of pixels to skip when searching for red circles
int SkipHoriz = 12;   // This number needs to be a multiple of 4...actually twice # to skip
int SkipVert  = 8;    // This is the actual number to skip in the vertical direction
if(HIGHRES == 1)
{
  SkipHoriz = 2*SkipHoriz;
  SkipVert  = 2*SkipVert;
}

// Number of pixels for search box around a discovered Red pixel
//   !!!!! Need to be EVEN numbers  !!!!!!!
int SearchBox = 16;   // 16 will create a 2*16+2*16+1 or 33x33 search box
if(HIGHRES == 1)
{
  SearchBox = 2*SearchBox;
}

//========= Conversion Variables
// Calibration Factors to convert pixels to meters
double YPixelsPerMeter = 128.62487;   // Pixels per meter in the Y direction
double XPixelsPerMeter = 133.98229;   // Pixels per meter in the X direction
if(HIGHRES == 1)
{
  YPixelsPerMeter = 2*YPixelsPerMeter;
  XPixelsPerMeter = 2*XPixelsPerMeter;
}

double RobotXMeters[Robots];       // X Position of the robot in meters
double RobotYMeters[Robots];       // Y Position of the robot in meters

//====== Theta Search Variables
double ThetaRadius = 23;     // Radius to search black/white semicircles  for Theta
if(HIGHRES == 1)
{
  ThetaRadius = 2*ThetaRadius;
}

int ThetaSearch[360][3];     // Array to store relative coords (x,y) of circular path
                             //  and corresponding Theta
int WhiteEdgeAngle = -179;  // Angle of last solid white pixel (initialized to starting angle)
int BlackEdgeAngle = -179;  // Angle of last solid black pixel (initialized to starting angle)
int LocationX, LocationY;   // Temporary storage of pixel location for lookup table

int WhiteFlag = -1;      // Flag for indication starting color in Theta search

//====== ID Number Search Variables
int RobotNumber;        // Counter to track individual robots
int RobotIndex;         // Holds base 10 version of ID number for storage in RobotID array

unsigned int TempID[Robots][4];    // Array to store Robot ID numbers as individual characters for
                                   //  easier manipulation
unsigned int RobotID[Robots];      // Array to store Robot ID as base 3 integers

// Array to store actual Robot ID numbers on the testbed
unsigned int ExistingIDs[ExistingRobots][4] = { 0, 2, 1, 0,
                        1, 0, 1, 1,
                        2, 1, 2, 1  };

// Used to store difference counters between measured ID and existing ID's
unsigned int IDDifference[ExistingRobots];

int MinDiff;          // Counts # of digits of difference between current & existing ID's
int MinIndex;         // Tracks location of smallest difference ID Number

double IDRadius = 12;
if(HIGHRES == 1)
{
  IDRadius = 2*IDRadius;
}

int IDSearch[360][2];

int DigitNotFound = 0;              // Flag to indicate an ID digit wasn't classified
```

```
//======== Image Loop Variables
//   !!!!! Need to be EVEN numbers  !!!!!!!
int StartOffsetX = Pad + 12;     // Loop variables for searching combined frames.
int StartOffsetY = Pad + 12;     //  We don't want to waste time searching the black
int EndOffsetX = 240 + Pad;      //  border (padding) of the combined image.
int EndOffsetY = 320 + Pad - CropLeft - CropRight - 12; //  These were found empirically.
if(HIGHRES == 1)
{
  StartOffsetX = Pad + 40;
  StartOffsetY = Pad + 40;
  EndOffsetX = 480 + Pad - 40;
  EndOffsetY = 640 + Pad - CropLeft - CropRight - 40;
}

// Dimensions from Calibration to stitch frames together:
// Initialize Calibration Variables. When I first calibrated these values, I did it
//  at the floor level, but the hats sit about 14" above the floor. Due to the cone
//  angle, the floor level calibrations were cropping pieces of the hat off when the
//  robot would move from one camera to another on the floor. Here I'm using all the
//  true pixels (I cropped out the artifacts) I can get to ensure an accurate measure-
//  -ment of the robots pose. Be careful if you change these, because each cameras uses
//  these values differently in the "for" loops below.

//   !!!!! Need to be EVEN numbers  !!!!!!!
// X and Y point of center of testbed (in the lower right corner of Cam 0)
int XCal0 = 240;                 // (old value)  223
int YCal0 = 320 - CropLeft - CropRight - 4; // (old value)  280
if(HIGHRES == 1)
{
  XCal0 = 480;
  YCal0 = 640 - CropLeft - CropRight - 8;
}

// X & Y point of center of testbed (in lower left corner of Cam 1)
int XCal1 = 240;                 // (old value)  222
int YCal1 = 10;                  // (old value)  15
if(HIGHRES == 1)
{
  XCal1 = 480;
  YCal1 = 20;
}

// X & Y point of center of testbed (in upper right corner of Cam 2)
int XCal2 = 0;                   // (old value) 11
int YCal2 = 320 - CropLeft - CropRight - 2; // (old value) 279
if(HIGHRES == 1)
{
  XCal2 = 0;
  YCal2 = 640 - CropLeft - CropRight - 4;
}

// X & Y point of center of testbed (in upper left corner of Cam 3)
int XCal3 = 2;                   // (old value) 12
int YCal3 = 10;                  // (old value) 19
if(HIGHRES == 1)
{
  XCal3 = 4;
  YCal3 = 20;
}

//======= Thresholding Variables
int YThreshTable[256];     // Y Lookup Table to do fast constant thresholding
int CbThreshTable[256];    // Cb Lookup Table to do fast constant thresholding
int CrThreshTable[256];    // Cr Lookup Table to do fast constant thresholding

int ThresholdResult;         // Holds result of bitwise AND operations for thresholding

int WhiteMask = 0x01;      // Masks to access color lookup result
int RedMask   = 0x02;
int BlueMask  = 0x04;
int GreenMask = 0x08;
int BlackMask = 0x10;
int Mask = 0;                        // Temporary Mask to use

int YFudge  = 0;          // Added variance for Y threshold levels
int CbFudge = 0;          // Added variance for Cb threshold levels
int CrFudge = 0;          // Added variance for Cr threshold levels

// Range of Threshold levels for each color and channel
//  Note: These can be set here, but they are later overwritten by data
//  from the ColorStats.txt file.  If you would like to set them manually,
//  you will need to comment out the 'Parsing Stats file' section below.
int RedYThreshLevelLow, RedYThreshLevelHigh;
int RedCbThreshLevelLow, RedCbThreshLevelHigh;
int RedCrThreshLevelLow, RedCrThreshLevelHigh;

int GreenYThreshLevelLow, GreenYThreshLevelHigh;
int GreenCbThreshLevelLow, GreenCbThreshLevelHigh;
int GreenCrThreshLevelLow, GreenCrThreshLevelHigh;

int BlueYThreshLevelLow, BlueYThreshLevelHigh;
```

```
int BlueCbThreshLevelLow, BlueCbThreshLevelHigh;
int BlueCrThreshLevelLow, BlueCrThreshLevelHigh;

int BlackYThreshLevelLow, BlackYThreshLevelHigh;
int BlackCbThreshLevelLow, BlackCbThreshLevelHigh;
int BlackCrThreshLevelLow, BlackCrThreshLevelHigh;

int WhiteYThreshLevelLow, WhiteYThreshLevelHigh;
int WhiteCbThreshLevelLow, WhiteCbThreshLevelHigh;
int WhiteCrThreshLevelLow, WhiteCrThreshLevelHigh;

int LowThreshold[5][3];   // Arrays to store lower and upper
int HighThreshold[5][3];  //  threshold bounds

int MinDataArray[5][3];   // To store min's of each color & channel
int MaxDataArray[5][3];   // To store max's of each color & channel

float MeanDataArray[5][3]; // To store mean of each color & channel
float StdDataArray[5][3];  // To store the Stdev of each color & chan.

//====== Data Storage File Streams
std::ofstream Tracking0;    // File to store Pose Information of the Robot
std::ofstream Tracking1;
std::ofstream Tracking2;

std::ofstream Grayfile;   // PGM grayscale image
std::ofstream RGBfile;    // RGB picture file in PPM format

//====== Communication Variables
std::string servAddress;                              // Temp address to send pose to robots
std::string servAddress0 = "192.168.1.111"; // Robot 1 (Adam) IP Address
std::string servAddress1 = "192.168.1.100"; // Robot 2 (Eve) IP Address
std::string servAddress2 = "192.168.1.101"; // Robot 3 (Abel) IP Address
char* echoString;              // Message to send to Robots
int echoStringLen;             // Length of string to echo = strlen(echoString)
unsigned short echoServPort = 7000;     // Port to Communicate to Robots
const int ECHOMAX = 255;                // Longest string to echo
char message[200];

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@ End Variables @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

{
// ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
// ^^^^^^^          Parse Command Line Arguments     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

if (argc != 2) {        // Test for correct number of parameters
  cerr << "Usage: " << argv[0] << " NumberOfIterationsToCapture\n\n" << endl;
  cerr << " For example, to measure 37 locations with the\n";
  cerr << " camera system type:  ./multitracker 37\n\n";
  exit(1);
}

CaptureIterations = atoi(argv[1]);


// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// $$$$$$$$$$$$  Parse Color Statistics File for Good Threshold Ranges  $$$$$$$$$$
// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

// Open the file stream for reading
std::ifstream ColorDataFile;  // This file holds statistics for each color
std::string FileName = "../ColorStats.txt";
ColorDataFile.open(FileName.c_str());

if(!ColorDataFile)
{
  std::cout << "Error Opening Color File\n";
  return EXIT_FAILURE;
}

// Read data from the text file into
std::string Label;          // Labels in data file for humans to read
std::string Value;          // Actual number that we need

// Read each color (Red, Green, Blue, Black, White)
for(color = 0; color < 5; color++)
{
    // Read each channel (Y, Cb, Cr)
  for(i = 0; i < 3; i++)
  {
      // Read Minimum value
    std::getline(ColorDataFile, Label, '=');
    std::getline(ColorDataFile, Value, ',');
    MinDataArray[color][i] = atoi(Value.c_str());

      // Read Mean value
    std::getline(ColorDataFile, Label, '=');
    std::getline(ColorDataFile, Value, ',');
    MeanDataArray[color][i] = atof(Value.c_str());
```

```
      // Read Standard Deviation value
    std::getline(ColorDataFile, Label, '=');
    std::getline(ColorDataFile, Value, ',');
    StdDataArray[color][i] = atof(Value.c_str());

      // Read Maximum value
    std::getline(ColorDataFile, Label, '=');
    std::getline(ColorDataFile, Value, ',');
    MaxDataArray[color][i] = atoi(Value.c_str());

  }
}

// Close Color Data File
ColorDataFile.close();

//$$$$$$$$$$$$$  End Parsing of Stats File  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

// Since we are using the Min and Max to set the range, I'm going to add a little
//  to each value just to capture a few more pixels that might not have been
//  present during the calibration.
int Fudge = 0;

for(color = 0; color < 5; color++)
{
  for(i = 0; i < 3; i++)
  {
    LowThreshold[color][i]  = MinDataArray[color][i] - Fudge;
    HighThreshold[color][i] = MaxDataArray[color][i] + Fudge;

    // Check boundaries (has to be [0 255])
    if(MinDataArray[color][i] < 0)    LowThreshold[color][i] = 0;
    if(MaxDataArray[color][i] > 255)  HighThreshold[color][i] = 255;
  }
}

//  // Here, we are going to use the Mean and the Standard deviation to set the
//  //  threshold ranges automatically. We will include a range centered at the
//  //  mean that covers a multiple of the standard deviation from the mean.
//  double StdevMultiplier = 4.0;
//
//  for(color = 0; color < 5; color++)
//  {
//          // "i" is each channel (YCbCr)
//          for(i = 0; i < 3; i++)
//          {
//                  LowThreshold[color][i] = (int)(MeanDataArray[color][i] -
//                  StdevMultiplier*(double)StdDataArray[color][i]);
//                  HighThreshold[color][i] = (int)(MeanDataArray[color][i] +
//                  StdevMultiplier*(double)StdDataArray[color][i]);
//
//                  // Check bounds of thresholds
//                  if(LowThreshold[color][i] < 0)          LowThreshold[color][i] = 0;
//                  if(HighThreshold[color][i] > 255)       HighThreshold[color][i] = 255;
//          }
//  }

// Copy values from DataArrays into Threshold Levels. This may seem inefficient
//  but this step allows us to change the method we use to specify the
//  threshold levels, without a major change to the code.
RedYThreshLevelLow      =  LowThreshold[0][0];
RedCbThreshLevelLow     =  LowThreshold[0][1];
RedCrThreshLevelLow     =  LowThreshold[0][2];

RedYThreshLevelHigh     =  HighThreshold[0][0];
RedCbThreshLevelHigh    =  HighThreshold[0][1];
RedCrThreshLevelHigh    =  HighThreshold[0][2];

GreenYThreshLevelLow    =  LowThreshold[1][0];
GreenCbThreshLevelLow   =  LowThreshold[1][1];
GreenCrThreshLevelLow   =  LowThreshold[1][2];

GreenYThreshLevelHigh   =  HighThreshold[1][0];
GreenCbThreshLevelHigh  =  HighThreshold[1][1];
GreenCrThreshLevelHigh  =  HighThreshold[1][2];

BlueYThreshLevelLow     =  LowThreshold[2][0];
BlueCbThreshLevelLow    =  LowThreshold[2][1];
BlueCrThreshLevelLow    =  LowThreshold[2][2];

BlueYThreshLevelHigh    =  HighThreshold[2][0];
BlueCbThreshLevelHigh   =  HighThreshold[2][1];
BlueCrThreshLevelHigh   =  HighThreshold[2][2];

BlackYThreshLevelLow    =  LowThreshold[3][0];
BlackCbThreshLevelLow   =  LowThreshold[3][1];
BlackCrThreshLevelLow   =  LowThreshold[3][2];

BlackYThreshLevelHigh   =  HighThreshold[3][0];
BlackCbThreshLevelHigh  =  HighThreshold[3][1];
BlackCrThreshLevelHigh  =  HighThreshold[3][2];
```

```
WhiteYThreshLevelLow    =  LowThreshold[4][0];
WhiteCbThreshLevelLow   =  LowThreshold[4][1];
WhiteCrThreshLevelLow   =  LowThreshold[4][2];

WhiteYThreshLevelHigh   =  HighThreshold[4][0];
WhiteCbThreshLevelHigh  =  HighThreshold[4][1];
WhiteCrThreshLevelHigh  =  HighThreshold[4][2];

//%%%%%%%% Debug - Print the levels used for thresholding
if(Debug > 0)
{
  std::cout << "Red:\n";
  std::cout << "Y\t" << RedYThreshLevelLow << "\t" << RedYThreshLevelHigh << "\n";
  std::cout << "Cb\t" << RedCbThreshLevelLow << "\t" << RedCbThreshLevelHigh << "\n";
  std::cout << "Cr\t" << RedCrThreshLevelLow << "\t" << RedCrThreshLevelHigh << "\n";

  std::cout << "Green:\n";
  std::cout << "Y\t" << GreenYThreshLevelLow << "\t" << GreenYThreshLevelHigh << "\n";
  std::cout << "Cb\t" << GreenCbThreshLevelLow << "\t" << GreenCbThreshLevelHigh << "\n";
  std::cout << "Cr\t" << GreenCrThreshLevelLow << "\t" << GreenCrThreshLevelHigh << "\n";

  std::cout << "Blue:\n";
  std::cout << "Y\t" << BlueYThreshLevelLow << "\t" << BlueYThreshLevelHigh << "\n";
  std::cout << "Cb\t" << BlueCbThreshLevelLow << "\t" << BlueCbThreshLevelHigh << "\n";
  std::cout << "Cr\t" << BlueCrThreshLevelLow << "\t" << BlueCrThreshLevelHigh << "\n";

  std::cout << "Black:\n";
  std::cout << "Y\t" << BlackYThreshLevelLow << "\t" << BlackYThreshLevelHigh << "\n";
  std::cout << "Cb\t" << BlackCbThreshLevelLow << "\t" << BlackCbThreshLevelHigh << "\n";
  std::cout << "Cr\t" << BlackCrThreshLevelLow << "\t" << BlackCrThreshLevelHigh << "\n";

  std::cout << "White:\n";
  std::cout << "Y\t" << WhiteYThreshLevelLow << "\t" << WhiteYThreshLevelHigh << "\n";
  std::cout << "Cb\t" << WhiteCbThreshLevelLow << "\t" << WhiteCbThreshLevelHigh << "\n";
  std::cout << "Cr\t" << WhiteCrThreshLevelLow << "\t" << WhiteCrThreshLevelHigh << "\n";
}

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@   Create a Thresholding Array for up to 32 colors   @@@@@@@@@@@@@@@@@@@
for(i = 0; i < 256; i++)
{
    YThreshTable[i]  = 0;        // Initialize arrays to zeros
    CbThreshTable[i] = 0;
    CrThreshTable[i] = 0;
}

// Place "1" in proper region for the Color Red for each channel (Y, Cb, Cr)
for(i = RedYThreshLevelLow; i <= RedYThreshLevelHigh; i++)
{
    YThreshTable[i] = YThreshTable[i] | RedMask;    // Y channel
}
for(i = RedCbThreshLevelLow; i <= RedCbThreshLevelHigh; i++)
{
    CbThreshTable[i] = CbThreshTable[i] | RedMask;
}
for(i = RedCrThreshLevelLow; i <= RedCrThreshLevelHigh; i++)
{
    CrThreshTable[i] = CrThreshTable[i] | RedMask;
}

// Place "1" in proper region for the Color Green for each channel (Y, Cb, Cr)
for(i = GreenYThreshLevelLow; i <= GreenYThreshLevelHigh; i++)
{
    YThreshTable[i] = YThreshTable[i] | GreenMask;    // Y channel
}
for(i = GreenCbThreshLevelLow; i <= GreenCbThreshLevelHigh; i++)
{
    CbThreshTable[i] = CbThreshTable[i] | GreenMask;
}
for(i = GreenCrThreshLevelLow; i <= GreenCrThreshLevelHigh; i++)
{
    CrThreshTable[i] = CrThreshTable[i] | GreenMask;
}

// Place "1" in proper region for the Color Blue for each channel (Y, Cb, Cr)
for(i = BlueYThreshLevelLow; i <= BlueYThreshLevelHigh; i++)
{
    YThreshTable[i] = YThreshTable[i] | BlueMask;    // Y channel
}
for(i = BlueCbThreshLevelLow; i <= BlueCbThreshLevelHigh; i++)
{
    CbThreshTable[i] = CbThreshTable[i] | BlueMask;
}
for(i = BlueCrThreshLevelLow; i <= BlueCrThreshLevelHigh; i++)
{
    CrThreshTable[i] = CrThreshTable[i] | BlueMask;
}

// Place "1" in proper region for the Color Black for each channel (Y, Cb, Cr)
for(i = BlackYThreshLevelLow; i <= BlackYThreshLevelHigh; i++)
{
```

```
    YThreshTable[i] = YThreshTable[i] | BlackMask;    // Y channel
}
for(i = BlackCbThreshLevelLow; i <= BlackCbThreshLevelHigh; i++)
{
    CbThreshTable[i] = CbThreshTable[i] | BlackMask;
}
for(i = BlackCrThreshLevelLow; i <= BlackCrThreshLevelHigh; i++)
{
    CrThreshTable[i] = CrThreshTable[i] | BlackMask;
}

// Place "1" in proper region for the Color White for each channel (Y, Cb, Cr)
for(i = WhiteYThreshLevelLow; i <= WhiteYThreshLevelHigh; i++)
{
    YThreshTable[i] = YThreshTable[i] | WhiteMask;    // Y channel
}
for(i = WhiteCbThreshLevelLow; i <= WhiteCbThreshLevelHigh; i++)
{
    CbThreshTable[i] = CbThreshTable[i] | WhiteMask;
}
for(i = WhiteCrThreshLevelLow; i <= WhiteCrThreshLevelHigh; i++)
{
    CrThreshTable[i] = CrThreshTable[i] | WhiteMask;
}
//@@@@@@@@  Finished Creating Threshold Table  @@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
}



//=======  Create Lookup Table for Orientation Search  ===========
for(a = 0; a < 360; a++)
{
  // Calculate X and Y coord moving CCWise starting at -179
  ThetaSearch[a][0] = -(int)(ThetaRadius*cos((M_PI*(double)a)/180));
  ThetaSearch[a][1] = -(int)(ThetaRadius*sin((M_PI*(double)a)/180));
  // Now store Theta. -179 is for converting angle to (-Pi,Pi]
  ThetaSearch[a][2] = a - 179;

  if(Debug > 10)
  {
    std::cout << "X= " << ThetaSearch[a][0] << " Y= " << ThetaSearch[a][1];
    std::cout << " T= " << ThetaSearch[a][2] << std::endl;
  }
}


//=======  Create Lookup Table for Robot ID Number Search  ===========
for(a = 0; a < 360; a++)
{
  // Calculate X and Y coord moving CCWise starting at 0
  IDSearch[a][0] = -(int)(IDRadius*cos((M_PI*(double)a)/180));
  IDSearch[a][1] = -(int)(IDRadius*sin((M_PI*(double)a)/180));

  if(Debug > 10)
  {
    std::cout << "ID: X= " << IDSearch[a][0] << " Y= " << IDSearch[a][1];
    std::cout << std::endl;
  }
}

if(Debug > 9)
{
  std::cout << "Ready to Track Robots\n";
}

//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$         Get Ready to Track Robots         $$$$$$$$$$$$$$$$$$$$$$$$

// Initialize variables for stitching together 4 frames
int HorizontalDiff0to2 = YCal0 - YCal2;
int VerticalDiff0to1   = XCal0 - XCal1;

// Setup the Frame Grabber and Cameras
// Open video device with YUV 422 format
Grab0.Init(CF_422, "/dev/video0", true);
Grab1.Init(CF_422, "/dev/video1", true);
Grab2.Init(CF_422, "/dev/video2", true);
Grab3.Init(CF_422, "/dev/video3", true);

// Set channel of multiplexor for input
Grab0.SetChannel("Composite0");
Grab1.SetChannel("Composite0");
Grab2.SetChannel("Composite0");
Grab3.SetChannel("Composite0");

// Video signal configure
Grab0.SetVideoSignal("ntsc");
Grab1.SetVideoSignal("ntsc");
Grab2.SetVideoSignal("ntsc");
Grab3.SetVideoSignal("ntsc");
```

```
Grab0.SetByteOrder(BYTE_ORDER_YUYV);
Grab1.SetByteOrder(BYTE_ORDER_YUYV);
Grab2.SetByteOrder(BYTE_ORDER_YUYV);
Grab3.SetByteOrder(BYTE_ORDER_YUYV);

if(Debug > 1)
{
  std::cout << "NewHeight = " << NewHeight << "\t";
  std::cout << "NewWidth = " << NewWidth << "\t";
  std::cout << "Padded Height = " << PaddedHeight << "\t";
  std::cout << "Padded Width = " << PaddedWidth << "\n";
}

// Open Text files to store pose information of the robot
if(Debug >= 0)
{
  Tracking0.open("Data/Tracking1011.txt", std::ios::out);
  Tracking1.open("Data/Tracking2121.txt", std::ios::out);
  Tracking2.open("Data/Tracking0210.txt", std::ios::out);
}


// Start grabbing
Grab0.Start();
Grab1.Start();
Grab2.Start();
Grab3.Start();

//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$    Begin Robot Tracking Loop     $$$$$$$$$$$$$$$

for(iter = 0; iter < CaptureIterations; iter++)
{

  std::cout << "Iter=" << iter << "\n";

  //#########################################################################
  //############    Crop Frames and Stitch Together    ######################
  //#########################################################################

  //@@@@ Upper Left Frame (Camera 0) @@@@@@@@@@@@@@@@@@@@@@@@*

  // Get frame from camera 0
  Grab0.Grab();

  // Quickly copy frame into array Frame for processing
  Grab0.CopyFrame((unsigned char *)&Frame[0]);

  // Move through the image  while cropping the right and left
  //   "artifacts" created by the frame grabber
  for(i = 0; i < NewHeight; i++)
  {
    Row = i*Width/2;

    for(j = 0; j < (NewWidth)/2; j++)
    {

      // Copy the packed YUYV values into new, padded array
      Padded0[(Pad+i)*PaddedWidth/2+(Pad/2)+j] = Frame[Row+j+CropLeft/2];

    }
  }


  //@@@@ Upper Right Frame (Camera 1) @@@@@@@@@@@@@@@@@@@@@@@@@@

  // Get frame from camera 1
  Grab1.Grab();

  // Quickly copy frame into array Frame1 for processing
  Grab1.CopyFrame((unsigned char *)&Frame[0]);

  // Move through the image while cropping the right and left
  //   "artifacts" created by the frame grabber
  for(i = 0; i < NewHeight; i++)
  {
    Row = i*Width/2;

    for(j = 0; j < (NewWidth)/2; j++)
    {

      // Copy the packed YUYV values into new, padded array
      Padded1[(Pad+i)*PaddedWidth/2+(Pad)/2+j] = Frame[Row+j+CropLeft/2];

    }
  }


  //@@@@ Lower Left Frame (Camera 2) @@@@@@@@@@@@@@@@@@@@@@@@@@

  // Get frame from camera 2
  Grab2.Grab();
```

```
// Quickly copy frame into array Frame2 for processing
Grab2.CopyFrame((unsigned char *)&Frame[0]);

// Move through the image while cropping the right and left
//   "artifacts" created by the frame grabber
for(i = 0; i < NewHeight; i++)
{
  Row = i*Width/2;

  for(j = 0; j < (NewWidth)/2; j++)
  {

    // Copy the packed YUYV values into new, padded array
    Padded2[(Pad+i)*PaddedWidth/2+(Pad)/2+j] = Frame[Row+j+CropLeft/2];

  }
}


//@@@@ Lower Right Frame (Camera 3) @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

// Get frame from camera 3
Grab3.Grab();

// Quickly copy frame into array Frame3 for processing
Grab3.CopyFrame((unsigned char *)&Frame[0]);

// Move through the image while cropping the right and left
//   "artifacts" created by the frame grabber
for(i = 0; i < NewHeight; i++)
{
  Row = i*Width/2;

  for(j = 0; j < (NewWidth)/2; j++)
  {

    // Copy the packed YUYV values into new, padded array
    Padded3[(Pad+i)*PaddedWidth/2+(Pad)/2+j] = Frame[Row+j+CropLeft/2];

  }
}

//############  End Cropping & Stitching  ######################################
//#################################################################################


//####### Save the Padded Image as a Grayscale PGM
// Note: This should really be converted to a function, but I was
//  unsuccessful at accomplishing this because of scope issues.
if(Debug > 0)
{
  // Open file stream for write operations to save PGM file
  Grayfile.open("Frames/padded0.pgm", std::ios::out);

  // Write Header for file
  Grayfile << "P2" << std::endl;
  Grayfile << PaddedWidth << " " << PaddedHeight << "\n";
  Grayfile << "255" << std::endl;

  // Copy Y pixels into file
  for(i = 0; i < PaddedHeight; i++)
  {
    for(j = 0; j < PaddedWidth; j = j + 2)
    {
      // Get 2 packed pixels from image
      pixels = Padded0[i*PaddedWidth/2+j/2];

      // Extract first intensity value
      Y = Y0(pixels);
      // Write to file
      Grayfile << Y << " ";
      // Extract 2nd intenisty value
      Y = Y1(pixels);
      // Write to file
      Grayfile << Y << " ";
    }
  }

  // Close file
  Grayfile.close();
}

//####### Save the Padded Image as a Grayscale PGM
// Note: This should really be converted to a function, but I was
//  unsuccessful at accomplishing this because of scope issues.
if(Debug > 0)
{
  // Open file stream for write operations to save PGM file
  Grayfile.open("Frames/padded1.pgm", std::ios::out);

  // Write Header for file
```

```
  Grayfile << "P2" << std::endl;
  Grayfile << PaddedWidth << " " << PaddedHeight << "\n";
  Grayfile << "255" << std::endl;

  // Copy Y pixels into file
  for(i = 0; i < PaddedHeight; i++)
  {
    for(j = 0; j < PaddedWidth; j = j + 2)
    {
      // Get 2 packed pixels from image
      pixels = Padded1[i*PaddedWidth/2+j/2];

      // Extract first intensity value
      Y = Y0(pixels);
      // Write to file
      Grayfile << Y << " ";
      // Extract 2nd intenisty value
      Y = Y1(pixels);
      // Write to file
      Grayfile << Y << " ";
    }
  }

  // Close file
  Grayfile.close();
}

//####### Save the Padded Image as a Grayscale PGM
// Note: This should really be converted to a function, but I was
//  unsuccessful at accomplishing this because of scope issues.
if(Debug > 0)
{
  // Open file stream for write operations to save PGM file
  Grayfile.open("Frames/padded2.pgm", std::ios::out);

  // Write Header for file
  Grayfile << "P2" << std::endl;
  Grayfile << PaddedWidth << " " << PaddedHeight << "\n";
  Grayfile << "255" << std::endl;

  // Copy Y pixels into file
  for(i = 0; i < PaddedHeight; i++)
  {
    for(j = 0; j < PaddedWidth; j = j + 2)
    {
      // Get 2 packed pixels from image
      pixels = Padded2[i*PaddedWidth/2+j/2];

      // Extract first intensity value
      Y = Y0(pixels);
      // Write to file
      Grayfile << Y << " ";
      // Extract 2nd intenisty value
      Y = Y1(pixels);
      // Write to file
      Grayfile << Y << " ";
    }
  }

  // Close file
  Grayfile.close();
}

//####### Save the Padded Image as a Grayscale PGM
// Note: This should really be converted to a function, but I was
//  unsuccessful at accomplishing this because of scope issues.
if(Debug > 0)
{
  // Open file stream for write operations to save PGM file
  Grayfile.open("Frames/padded3.pgm", std::ios::out);

  // Write Header for file
  Grayfile << "P2" << std::endl;
  Grayfile << PaddedWidth << " " << PaddedHeight << "\n";
  Grayfile << "255" << std::endl;

  // Copy Y pixels into file
  for(i = 0; i < PaddedHeight; i++)
  {
    for(j = 0; j < PaddedWidth; j = j + 2)
    {
      // Get 2 packed pixels from image
      pixels = Padded3[i*PaddedWidth/2+j/2];

      // Extract first intensity value
      Y = Y0(pixels);
      // Write to file
      Grayfile << Y << " ";
      // Extract 2nd intenisty value
      Y = Y1(pixels);
      // Write to file
      Grayfile << Y << " ";
```

```
    }
  }

  // Close file
  Grayfile.close();
}

//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&&&&&&  Search for Robots  &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&


// Initialize Tracking Variables
RedCenterX  = 0;
RedCenterY  = 0;
RedMass     = 0;

RobotNumber = 0;

// Clear the robot ID's... if a 5 shows up in the output, then we know a certain
//  part of the color ID band wasn't classified.
for(i = 0; i < Robots; i++)
{
  TempID[i][0] = 5;
  TempID[i][1] = 5;
  TempID[i][2] = 5;
  TempID[i][3] = 5;
}

// Initialize robot ID's in the integer array to impossible value (max ID is 81)
for(i = 0; i < Robots; i++)
{
  RobotID[i] = 90;
}

if(Debug > 2)
{
  std::cout << "\nStarting search for robots\n";
}

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@  Search for Red Circles in all padded frames  @@@@@@@@@@@@@@
for(Camera = 0; Camera < 4; Camera++)
{

  // Select correct camera frame to search
  switch (Camera)
  {
    case 0:
      FramePtr = &Padded0[0];
      break;
    case 1:
      FramePtr = &Padded1[0];
      break;

    case 2:
      FramePtr = &Padded2[0];
      break;

    case 3:
      FramePtr = &Padded3[0];
      break;
  }

  // Initialize RGB thresholded picture values to White
  if(Debug > 0)
  {
    // Initialize to white
    for(i = 0; i < PaddedHeight*PaddedWidth; i++)
    {
      ThreshPic[i] = RGB(255, 255, 255);
    }
  }


  for(i = StartOffsetX; i < EndOffsetX; i = i + SkipVert)
  {
    for(j = StartOffsetY; j < EndOffsetY; j = j + SkipHoriz/2)
    {
      if(Debug > 10)
      {
        std::cout << "i = " << i << " j = " << j << std::endl;
      }

      // Get 2 compressed Pixels from Framegrabber
      //pixels = Padded0[i*PaddedWidth/2+j/2];
      pixels = *(FramePtr+i*PaddedWidth/2+j/2);

      //%%%%%%%%  Need to include an if statement to test whether
      //%%%%%%%%   the program needs an even or an odd pixel. Then
      //%%%%%%%%   it should unpack the appropriate one.
```

```
// If the pixel requested is even
if( j % 2 == 0)
{
  // Extract even numbered Intensity pixel -> Y0
  Y = Y0(pixels);
}
//...of if the number is odd
else if( j % 2 == 1)
{
  // Extract odd numbered Intensity pixel -> Y1
  Y = Y1(pixels);
}

// Extract the color component of the pixels
Cb = CB(pixels);
Cr = CR(pixels);

//#!!!!!! Need to check if pixel is part of previously found robot
if(Cr == 255)
{
  // This pixel has already been found to be red b/c
  //  the cameras have never produced pure red and likely
  //  never will. So, we use that to our advantage here
  //  and move on to next pixel
  continue;
}

// Threshold the image using lookup tables and bitwise AND statements
ThresholdResult = YThreshTable[Y] & CbThreshTable[Cb] & CrThreshTable[Cr];

if(ThresholdResult & RedMask)
{

  if(Debug > 1)
  {
    std::cout << "\nDetected Red at " << i << ", " << j << "\n";

  }

  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  //@@@@@@@@@@@@@ Begin Search for Robot's Center @@@@@@@@@@@@@@@@@@@
  // If we found a red pixel, then do a refined local search for
  //  the center of this robot
  for(f = -SearchBox; f <= SearchBox; f++)
  {
    for(g = -SearchBox; g <= SearchBox; g = g + 2)
    {

      //@!!!!!! We need to start on an EVEN pixel to insure accuracy
      //@!!!!!!  of center of mass calculation !!!!!!!!
      // If this is the first iteration....
      if(g == -SearchBox)
      {
        // If the starting Y coord is odd...
        if(-g % 2 == 1)
        {
          //scoot over 1 pixel to right to make it even
          g = -SearchBox + 1;
        }
      }

      // Get 2 compressed Pixels from Framegrabber
      //pixels = Padded0[(i+f)*PaddedWidth/2+(j+g)/2];
      pixels = *(FramePtr+(i+f)*PaddedWidth/2+(j+g)/2);

      // Extract first pixel from packed pixel
      Y  = Y0(pixels);
      Cb = CB(pixels);
      Cr = CR(pixels);

      // Threshold the image using lookup tables and bitwise AND statements
      ThresholdResult = YThreshTable[Y] & CbThreshTable[Cb] & CrThreshTable[Cr];

      if(ThresholdResult & RedMask)
      {

        // Keep track of Red mass
        // Sum all x coordinates
        RedCenterX = RedCenterX + (i + f);
        // Sum all y coordinates
        RedCenterY = RedCenterY + (j + g);
        RedMass++;                    // Track mass of Red pixels

        // Store thresholded result as pure YUYV Red Pixels
        //Padded0[(i+f)*PaddedWidth/2+(j+g)/2] = YUYV(76, 85, 76, 255);
        *(FramePtr+(i+f)*PaddedWidth/2+(j+g)/2) = YUYV(76, 85, 76, 255);

        if(Debug > 10)
        {
          std::cout << "\nTracking#" << RobotNumber << " and found red pix at ";
          std::cout << i+f << " " << j+g;
        }
```

```
      if(Debug > 0)
      {
        // Also store in Thresholded image as RED
        ThreshPic[(i+f)*PaddedWidth+(j+g)] = RGB(255, 0, 0);
      }

    }
    else
    {

      if(Debug > 0)
      {
        // Pixel was unclassified, so set to Black
        ThreshPic[(i+f)*PaddedWidth+(j+g)] = RGB(0, 0, 0);
      }

    }

    // Extract second pixel from packed pixel
    Y  = Y1(pixels);
    Cb = CB(pixels);
    Cr = CR(pixels);

    // Threshold the image using lookup tables and bitwise AND statements
    ThresholdResult = YThreshTable[Y] & CbThreshTable[Cb] & CrThreshTable[Cr];

    if(ThresholdResult & RedMask)
    {

      // Keep track of Red mass
      // Sum all x coordinates
      RedCenterX = RedCenterX + (i + f);
      // Sum all y coordinates (remember to add 1 b/c it's 2nd pixel)
      RedCenterY = RedCenterY + (j + g + 1);
      RedMass++;                    // Track mass of Red pixels

      // Store thresholded result as pure YUYV Red Pixels
      //Padded0[(i+f)*PaddedWidth/2+(j+g)/2] = YUYV(76, 85, 76, 255);
      *(FramePtr+(i+f)*PaddedWidth/2+(j+g)/2) = YUYV(76, 85, 76, 255);

      // For fastest execution
      if(Debug == 0)
      {
        continue;
      }

      if(Debug > 10)
      {
        std::cout << "\nTracking#" << RobotNumber << "_and_found_red_pix_at_";
        std::cout << i+f << "_" << j+g;
      }

      if(Debug > 0)
      {
        // Also store in Thresholded image as RED
        ThreshPic[(i+f)*PaddedWidth+(j+g+1)] = RGB(255, 0, 0);
      }
    }
    else
    {

      if(Debug > 0)
      {
        // Pixel was unclassified, so set to Black
        ThreshPic[(i+f)*PaddedWidth+(j+g+1)] = RGB(0, 0, 0);
      }

    }

  }
}
//@@@@@@@@@@@@@  End Search for Robot's Center  @@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

// Make sure that a red spot was actually a robot using a "size filter"
if(RedMass < MinRedSize)
{
  if(Debug > 0)
  {
    std::cout << "Got_Red_noise_at_X=_" << i << "_Y=_" << j << "\n";
  }

  // Go to next sparse search pixel...this was some speck of noise
  continue;
}

// Calculate Center of Mass for the Red Circle (a robot)
RedCenterX = RedCenterX/RedMass;
RedCenterY = RedCenterY/RedMass;

if(Debug > 0)
```

```
{
  // Put a Green Dot at center of red mass in ThreshPic
  ThreshPic[(int)RedCenterX*PaddedWidth+(int)RedCenterY] = RGB(0, 255, 0);
}

// Store center in RobotX-Y arrays
RobotX[RobotNumber] = RedCenterX;
RobotY[RobotNumber] = RedCenterY;

if(Debug > 6)
{
  std::cout << "\nReady to find Theta\n";
}

//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$$   Begin Theta Search  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// Now, find the Orientation by circling from the center
//  at a specified radius looking for the White/Black interface
for(a = 0; a < 360; a = a + DegreeStep)
{

  // Use a lookup table for the circular coordinate route
  LocationX = (int)RobotX[RobotNumber] + ThetaSearch[a][0];
  LocationY = (int)RobotY[RobotNumber] + ThetaSearch[a][1];

  if(Debug > 10)
  {
    std::cout << "\nLocationX and Y = " << LocationX << " ";
    std::cout << LocationY;
  }

  // Get 2 compressed Pixels from Framegrabber
  //pixels = Padded0[LocationX*PaddedWidth/2+LocationY/2];
  pixels = *(FramePtr+LocationX*PaddedWidth/2+LocationY/2);

  //%%%%%%%%  Need to include an if statement to test whether
  //%%%%%%%%   the program needs an even or an odd pixel. Then
  //%%%%%%%%   it unpacks the appropriate one.

  // If the pixel requested is even
  if( LocationY % 2 == 0)
  {
    // Extract even numbered Intensity pixel -> Y0
    Y = Y0(pixels);
  }
  //...of if the number is odd
  else if( LocationY % 2 == 1)
  {
    // Extract odd numbered Intensity pixel -> Y1
    Y = Y1(pixels);
  }

  // Extract the color component of the pixels
  Cb = CB(pixels);
  Cr = CR(pixels);

  // Threshold the image using lookup tables and bitwise AND statements
  ThresholdResult = YThreshTable[Y] & CbThreshTable[Cb] & CrThreshTable[Cr];

  // Is this the first pixel? Or was the 1st misclassified?
  if(a == 0 | WhiteFlag == -1)
  {
    // Did we start in White? Then look for black
    if(ThresholdResult & WhiteMask)
    {
      WhiteFlag = 1;
      WhiteEdgeAngle = ThetaSearch[a][2];
      if(Debug > 0)
      {
        ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(255, 255, 0);
      }
      continue;   // Try next pixel, since this is already classified
    }
    else if(ThresholdResult & BlackMask)
    {
      WhiteFlag = 0;
      BlackEdgeAngle = ThetaSearch[a][2];
      if(Debug > 0)
      {
        ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(75, 75, 75);
      }
      continue;   // Try next pixel, since this is already classified
    }
    else
    {
      // We weren't able to classify the pixel as white or black
      WhiteFlag = -1;

      if(Debug > 0)
      {
        std::cout << "\n" << a+1 << " pixel unclassified ";
        std::cout << "when searching for Theta\n";
```

```
      ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(128, 0, 128);
    }

    continue;   // Move to next pixel to try and classify it
  }
}

// If starting pixel was white, look for black edge
if(WhiteFlag == 1)
{

  // Is this pixel White?
  if(ThresholdResult & WhiteMask)
  {
    // Store this angle for more accurate Theta results
    WhiteEdgeAngle = ThetaSearch[a][2];

    if(Debug > 0)
    {
      ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(128, 128, 128);
    }

  }
  // Or is this pixel Black?
  else if(ThresholdResult & BlackMask)
  {
    // Average angles from solid black and solid white
    //  We add 180 to theta b/c we started in white
    Theta[RobotNumber] = 180 + (double)(ThetaSearch[a][2] + WhiteEdgeAngle)/2;

    // Check proper bounds for theta (-Pi,Pi]
    if(Theta[RobotNumber] >  180) Theta[RobotNumber] = Theta[RobotNumber] - 360;
    if(Theta[RobotNumber] < -179) Theta[RobotNumber] = Theta[RobotNumber] + 360;

    if(Debug > 0)
    {
      // Put Green Dot at black/white interface
      ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(0, 255, 0);

      //std::cout << "Found Theta\n";
    }

    // We found the Theta, so exit the search loop
    break;
  }
  // ....or Pixel wasn't classified at all
  else
  {
    if(Debug > 0)
    {
      // Put purple dot for "not classified"
      ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(128, 0, 128);
    }
  }

}
// If starting pixel was black, look for white edge
else if(WhiteFlag == 0)
{

  // Is this pixel black?
  if(ThresholdResult & BlackMask)
  {
    // Store this angle for more accurate Theta results
    BlackEdgeAngle = ThetaSearch[a][2];

    if(Debug > 0)
    {
      ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(128, 128, 128);
    }

  }

  // Or is this pixel White?
  else if(ThresholdResult & WhiteMask)
  {
    // Average angles from solid black and solid white
    Theta[RobotNumber] = (double)(ThetaSearch[a][2] + BlackEdgeAngle)/2;

    // Check proper bounds for theta (-Pi,Pi]
    if(Theta[RobotNumber] >  180) Theta[RobotNumber] = Theta[RobotNumber] - 360;
    if(Theta[RobotNumber] < -179) Theta[RobotNumber] = Theta[RobotNumber] + 360;

    if(Debug > 0)
    {
      // Put Green Dot at black/white interface
      ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(0, 255, 0);

      //std::cout << "Found Theta\n";
    }

    // We found the Theta, so exit the search loop
```

```
      break;
    }
    // ....or Pixel wasn't classified at all
    else
    {
      if(Debug > 0)
      {
        // Put purple dot for "not classified"
        ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(128, 0, 128);
      }
    }

  }

}
//$$$$$$$$$$$  End Theta Search  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


//#################################################################
//############  Begin ID Number Search  ###########################
// Now, find the robot identifier number by moving around the
//  other radius. Searching for 4 digits with White=0, Blue=1
//  and Green=2
int digit = 0;
a = 0;

while(digit < 4)
{

  // Start Looking at first digit on front of robot
  //   +179 is to make index a positive value for lookup table
  //   -25 is to try and search most of the quarter ring in case
  //      of mis-classifications = greater room for error
  int AngleIndex = 179 + (int)Theta[RobotNumber] + digit*90 - 25 + a;

  // Check bounds of Angle Index...should be between 0 and 359
  if(AngleIndex > 360)     AngleIndex = AngleIndex - 360;
  if(AngleIndex < 0)               AngleIndex = AngleIndex + 360;

  // Use a lookup table for the circular coordinate route
  LocationX = (int)RobotX[RobotNumber] + IDSearch[AngleIndex][0];
  LocationY = (int)RobotY[RobotNumber] + IDSearch[AngleIndex][1];

  // Get 2 compressed Pixels from Framegrabber
  //pixels = Padded0[LocationX*PaddedWidth/2+LocationY/2];
  pixels = *(FramePtr+LocationX*PaddedWidth/2+LocationY/2);

  //%%%%%%%%  Need to include an if statement to test whether
  //%%%%%%%%   the program needs an even or an odd pixel. Then
  //%%%%%%%%   it unpacks the appropriate one.

  // If the pixel requested is even
  if( LocationY % 2 == 0)
  {
    // Extract even numbered Intensity pixel -> Y0
    Y = Y0(pixels);
  }
  //...of if the number is odd
  else if( LocationY % 2 == 1)
  {
    // Extract odd numbered Intensity pixel -> Y1
    Y = Y1(pixels);
  }

  // Extract the color component of the pixels
  Cb = CB(pixels);
  Cr = CR(pixels);

  // Threshold the image using lookup tables and bitwise AND statements
  ThresholdResult = YThreshTable[Y] & CbThreshTable[Cb] & CrThreshTable[Cr];

  if(ThresholdResult & WhiteMask)
  {
    // Store the ID digit
    TempID[RobotNumber][digit] = 0;
    // Go to next quarter circle
    digit++;
    // Clear next pixel counter
    a = 0;

    if(Debug > 0)
    {
      // Set pixel to Yellow for easier viewing
      ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(255,217,0);
    }
  }
  else if(ThresholdResult & BlueMask)
  {
    // Store the ID digit
    TempID[RobotNumber][digit] = 1;
    // Go to next quarter circle
```

```
      digit++;
      // Clear next pixel counter
      a = 0;

      if(Debug > 0)
      {
        // Set pixel to Blue
        ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(0,0,255);
      }
    }
    else if(ThresholdResult & GreenMask)
    {
      // Store the ID digit
      TempID[RobotNumber][digit] = 2;
      // Go to next quarter circle
      digit++;
      // Clear next pixel counter
      a = 0;

      if(Debug > 0)
      {
        // Set pixel to Green
        ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(0,255,0);
      }
    }
    else
    {
      if(Debug > 0)
      {
        //std::cout << "ID Pixel for digit " << digit;
        //std::cout << " not classified. Moving to next pixel\n";

        // Set Pixel to Purple
        ThreshPic[LocationX*PaddedWidth+LocationY] = RGB(128,0,128);
      }

      // See if the next pixel will classify
      a++;

      // Limit how many extra pixels to search
      if(a > 60)
      {
        // Give up and move to next digit
        // Store the ID digit
        TempID[RobotNumber][digit] = 4;      // Incorrect value
        digit++;
        a = 0;
        // Set Flag that digit wasn't found
        DigitNotFound = 1;
      }
    }

  }


  //^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  //^^^^^^^^^  Error Correction of Robot ID Numbers  ^^^^^^^^^^^^^^^
  if(ErrorCorrection == 1)
  {
    // Clear ID Differences
    for(r = 0; r < ExistingRobots; r++)      { IDDifference[r] = 0; }

    // Initialize to impossible values
    MinDiff = 10;
    MinIndex = -1;

    // Compare Measured ID # to Existing ID #'s
    for(r = 0; r < ExistingRobots; r++)
    {
      // Compare each digit of ID # to Existing ID # digits
      for(digit = 0; digit < 4; digit++)
      {
        // Test if Current ID digit and an Exisiting ID digit DOESN'T match
        if( ExistingIDs[r][digit] != TempID[RobotNumber][digit] )
        {
          //....These digits didn't match
          IDDifference[r]++;

        }
      }

      // Do we have a perfect match with an existing ID?
      if(IDDifference[r] == 0)
      {
        // Store the location of perfect match
        MinIndex = r;
        //...then stop the search
        break;
      }
      //.... There was not a perfect match for this Existing ID
      else if(IDDifference[r] < MinDiff)
      {
```

```
      //...or keep track of the smallest difference
      MinDiff = IDDifference[r];
      // and the ID location of the smallest difference
      MinIndex = r;
    }
  }

  // Calculate base 10 equivalent of robot ID for storage in array
  RobotIndex = TempID[RobotNumber][0]*27 + TempID[RobotNumber][1]*9 +
      TempID[RobotNumber][2]*3 + TempID[RobotNumber][3];

  // Did we already find this robot? (90 was the initialization value)
  if(RobotID[RobotIndex] == 90)
  {
    // If not, store the base 3 integer ID (for easier human reading)
    // Convert RobotID array values to single Base 3 (sort of) integer
    //   I say 'sort of' because of the error values of 4 or 5 the might show up
    RobotID[RobotIndex] = ExistingIDs[MinIndex][0]*1000 + ExistingIDs[MinIndex][1]*100
        + ExistingIDs[MinIndex][2]*10 + ExistingIDs[MinIndex][3];
  }
  else
  {
    // We already found this robot, so don't send out pose by
    //  setting this flag
    AlreadyFound = 1;

    if(Debug > 0)
    {
      std::cout << "Already Found " << TempID[RobotNumber][0]
          << TempID[RobotNumber][1] << TempID[RobotNumber][2]
          << TempID[RobotNumber][3] << "\n";
    }

  }

}
else if( ErrorCorrection != 1)
{

  // Calculate base 10 equivalent of robot ID for storage in array
  RobotIndex = TempID[RobotNumber][0]*27 + TempID[RobotNumber][1]*9 +
      TempID[RobotNumber][2]*3 + TempID[RobotNumber][3];

  // Did we already find this robot? (90 was the initialization value)
  if(RobotID[RobotIndex] == 90)
  {
    // If not, store the base 3 integer ID (for easier human reading)
    RobotID[RobotIndex] = TempID[RobotNumber][0]*1000 + TempID[RobotNumber][1]*100
        + TempID[RobotNumber][2]*10 + TempID[RobotNumber][3];
  }
  else
  {
    // We already found this robot, so don't send out pose by
    //  setting this flag
    AlreadyFound = 1;

    if(Debug > 0)
    {
      std::cout << "Already Found " << TempID[RobotNumber][0]
          << TempID[RobotNumber][1] << TempID[RobotNumber][2]
          << TempID[RobotNumber][3] << "\n";
    }

  }
}

//###########  End ID Number Search  ##########################
//#################################################################

//$$$$$$$$ Correct center of robot for camera offsets from origin
//$$$$$$$$  and convert pixels to meters
switch(Camera)
{
  // Camera 0
  case 0:
    // No offset for camera 0
    // Just convert pixels to meters and store
    RobotXMeters[RobotNumber] = RobotX[RobotNumber]/XPixelsPerMeter;
    RobotYMeters[RobotNumber] = RobotY[RobotNumber]/YPixelsPerMeter;
    break;

  // Camera 1
  case 1:
    // Adjust the offset
    RobotX[RobotNumber] = RobotX[RobotNumber] + VerticalDiff0to1;
    RobotY[RobotNumber] = RobotY[RobotNumber] + YCal0 - YCal1;

    // Also convert pixels to meters and store
    RobotXMeters[RobotNumber] = RobotX[RobotNumber]/XPixelsPerMeter;
    RobotYMeters[RobotNumber] = RobotY[RobotNumber]/YPixelsPerMeter;
    break;
```

```
    //Camera 2
    case 2:
      // Adjust the offset
      RobotX[RobotNumber] = RobotX[RobotNumber] + XCal0 − XCal2;
      RobotY[RobotNumber] = RobotY[RobotNumber] + HorizontalDiff0to2;

      // Also convert pixels to meters and store
      RobotXMeters[RobotNumber] = RobotX[RobotNumber]/XPixelsPerMeter;
      RobotYMeters[RobotNumber] = RobotY[RobotNumber]/YPixelsPerMeter;
      break;

    case 3:
      // Adjust the offset
      RobotX[RobotNumber] = RobotX[RobotNumber] + XCal0 − XCal3;
      RobotY[RobotNumber] = RobotY[RobotNumber] + YCal0 − YCal3;

      // Also convert pixels to meters and store
      RobotXMeters[RobotNumber] = RobotX[RobotNumber]/XPixelsPerMeter;
      RobotYMeters[RobotNumber] = RobotY[RobotNumber]/YPixelsPerMeter;
      break;

  }

  if( AlreadyFound != 1)
  {

    //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    //@@@@@@   Print out Robot Information   @@@@@@@@@@@@@@@@@@@@@@@@
    if(Debug > −1)
    {

      // Print ID number with temporary ID
      std::cout << "Robot #" << RobotNumber << " = " << TempID[RobotNumber][0];
      std::cout << TempID[RobotNumber][1] << TempID[RobotNumber][2];
      std::cout << TempID[RobotNumber][3];

      // Also Print Pose (X, Y, Theta)
      std::cout << " is at X= " << RobotX[RobotNumber];
      std::cout << " = " << RobotXMeters[RobotNumber] << "m";
      std::cout << " Y= " << RobotY[RobotNumber];
      std::cout << " = " << RobotYMeters[RobotNumber] << "m";
      std::cout << " Theta= " << Theta[RobotNumber] << "\n";

    }

    // Store movement of robots in file
    if(Debug >= 0)
    {
      switch (RobotID[RobotIndex])
      {
        case 1011:
          Tracking0 << 1011 << "\t";
          Tracking0 << RobotX[RobotNumber]  << "\t";
          Tracking0 << RobotXMeters[RobotNumber] << "\t";
          Tracking0 << RobotY[RobotNumber]  << "\t";
          Tracking0 << RobotYMeters[RobotNumber] << "  \t";
          Tracking0 << Theta[RobotNumber]   << "\n";
          servAddress = servAddress0;
          break;

        case 2121:
          Tracking1 << 2121 << "\t";
          Tracking1 << RobotX[RobotNumber]  << "\t";
          Tracking1 << RobotXMeters[RobotNumber] << "\t";
          Tracking1 << RobotY[RobotNumber]  << "\t";
          Tracking1 << RobotYMeters[RobotNumber] << "  \t";
          Tracking1 << Theta[RobotNumber]   << "\n";
          servAddress = servAddress1;
          break;

        case 210:
          Tracking2 << 210 << "\t";
          Tracking2 << RobotX[RobotNumber]  << "\t";
          Tracking2 << RobotXMeters[RobotNumber] << "\t";
          Tracking2 << RobotY[RobotNumber]  << "\t";
          Tracking2 << RobotYMeters[RobotNumber] << "  \t";
          Tracking2 << Theta[RobotNumber]   << "\n";
          servAddress = servAddress2;
          break;

        default:
          std::cout << "One Robot ID " << RobotID[RobotIndex] << " wasn't";
          std::cout << " classified\nThis ID should be added to switch/case";
          std::cout << " statement to tell the \nprogram what file to write";
          std::cout << " the tracking data to.\n";
      }

      //@@@@@@ Send Pose Info to Robots @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
      // Convert Pose numbers to string for transmission
      std::sprintf(message, "%f %f %f", RobotXMeters[RobotNumber],
            RobotYMeters[RobotNumber], M_PI*(Theta[RobotNumber])/180);
      echoString = message;
```

```
          echoStringLen = strlen(echoString);

          try {
            UDPSocket sock;

            // Send the string to the server
            sock.sendTo(echoString, echoStringLen, servAddress, echoServPort);

            // Destructor closes the socket

          } catch (SocketException &e) {
            cerr << e.what() << endl;
            exit(1);
          }

        }

      }

      //#####################################################
      //####  Clean up and Reset variables for next robot

      // Clear Tracking variables for next robot
      RedCenterX  = 0;
      RedCenterY  = 0;
      RedMass     = 0;

      // Increment Robot Number for the next found robot
      RobotNumber++;

      // Clear White Flag
      WhiteFlag = -1;

      // Clear ID Digit not Found Flag
      DigitNotFound = 0;

      // Clear Already Found flag
      AlreadyFound = 0;

    }
    else
    {

      // For fastest execution
      if(Debug == 0)
      {
        continue;
      }

      // Store in Thresholded Picture as Black
      ThreshPic[i*PaddedWidth+j] = RGB(0, 0, 0);

    }

  }
}


//#####################################################################
//      Save the Thresholded Image as an RGB PPM
// Note: This should really be converted to a function, but I was
//  unsuccessful at accomplishing this because of scope issues.
if(Debug > 0)
{
  // Open file stream for write operations to save PGM file
  switch(Camera)
  {
    case 0:
      RGBfile.open("Frames/threshpic0.ppm", std::ios::out);
      break;

    case 1:
      RGBfile.open("Frames/threshpic1.ppm", std::ios::out);
      break;

    case 2:
      RGBfile.open("Frames/threshpic2.ppm", std::ios::out);
      break;

    case 3:
      RGBfile.open("Frames/threshpic3.ppm", std::ios::out);
      break;

  }

  // Write Header for file
  RGBfile << "P3" << std::endl;
  RGBfile << PaddedWidth << "␣" << PaddedHeight << "\n";
  RGBfile << "255" << std::endl;

  // Copy Y pixels into file
  for(i = 0; i < PaddedHeight; i++)
```

```
            {
              for(j = 0; j < PaddedWidth; j++)
              {
                // Get 2 packed pixels from image
                rgbpixel = ThreshPic[i*PaddedWidth+j];

                // Extract channel values
                R = RED(rgbpixel);
                G = GREEN(rgbpixel);
                B = BLUE(rgbpixel);

                // Write to file
                RGBfile << R << " " << G << " " << B << " ";

              }
            }

            // Close file
            RGBfile.close();
          }

        }


      }  // End Tracking

      if(Debug >= 0)
      {
        // Close Pose Tracking Text files
        Tracking0.close();
        Tracking1.close();
        Tracking2.close();
      }

      // Stop Grabbing Frames
      Grab0.Stop();
      Grab1.Stop();
      Grab2.Stop();
      Grab3.Stop();

      return 0;
}
```

APPENDIX III

Color Threshold Calibration Data

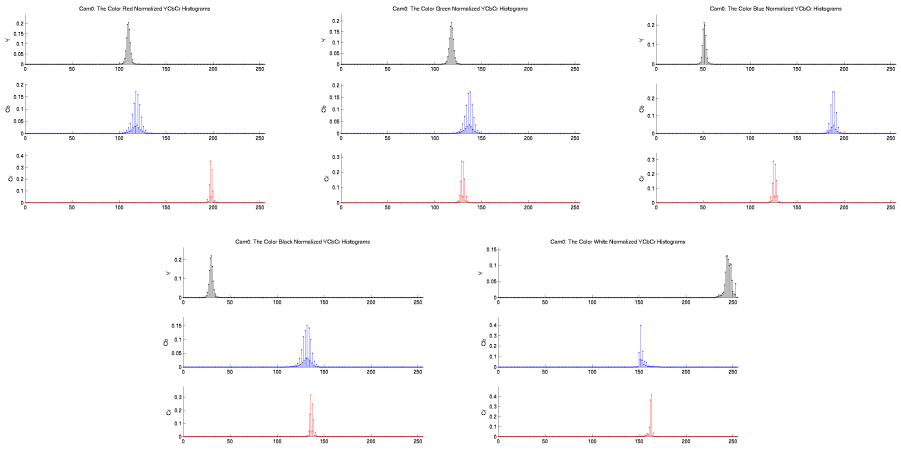

FIGURE 59 – Camera 0: From the LEFT: YCbCr Histograms of Red, Green, Blue, Black and White.

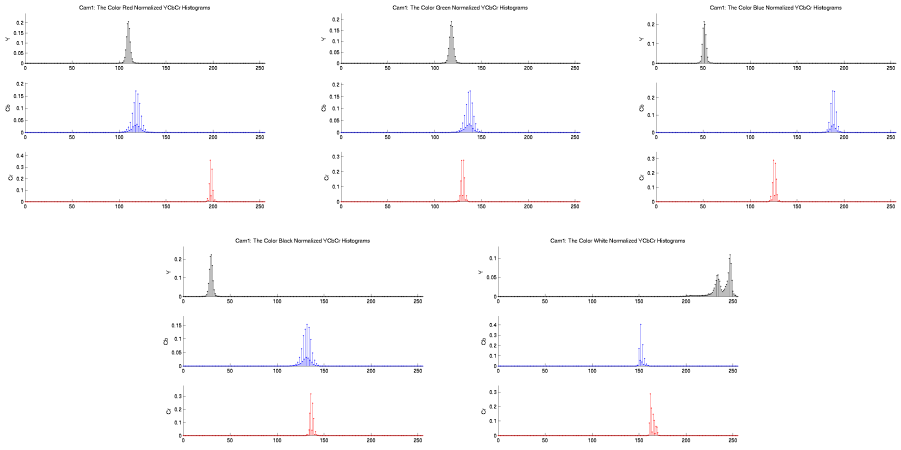

FIGURE 60 – Camera 1: From the LEFT: YCbCr Histograms of Red, Green, Blue, Black and White.

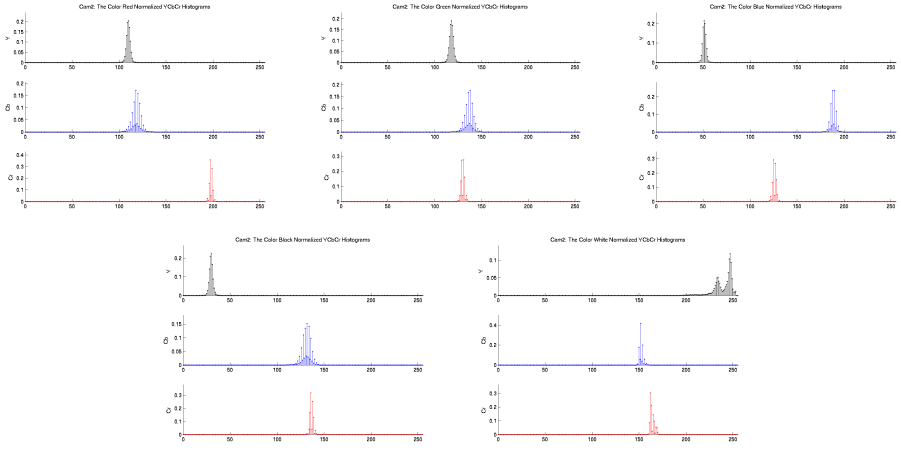

FIGURE 61 – Camera 2: From the LEFT: YCbCr Histograms of Red, Green, Blue, Black and White.

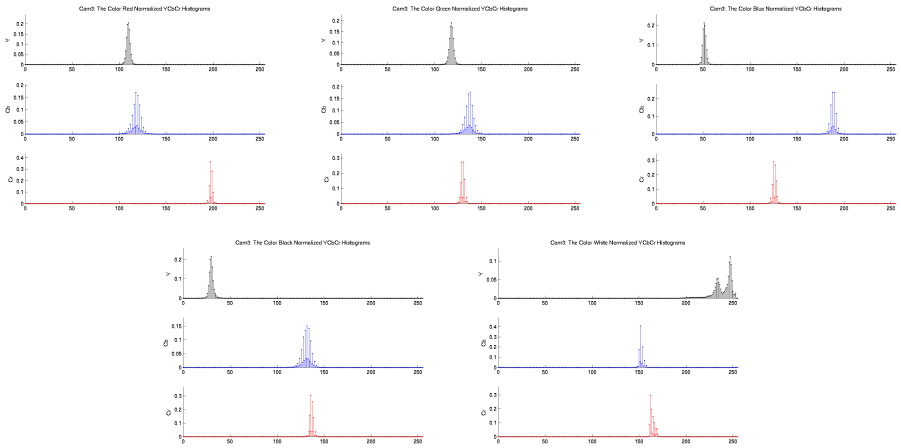

FIGURE 62 – Camera 3: From the LEFT: YCbCr Histograms of Red, Green, Blue, Black and White.

APPENDIX IV

Color Threshold Calibration Sampling Program

```
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$
//$$              Color Calibration Program
//$$                by Travis Riggs
//$$                  10/12/2006
//$$
//$$     This program grabs a video frame from a camera through video
//$$     for linux frame grabber. It uses some of the library functions
//$$     to communicate with the camera. Note that those functions are
//$$     written in C, but this main program is written in C++.
//$$
//$$     This program is intended to be used to calculate and store
//$$     the pixel values for a single color at a time. It writes
//$$     the YCbCr pixel values to a text file to be analyzed by
//$$     Matlab or Octave later.
//$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
//$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

//%%%%%%%%%%%%%   Header Files   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#include <iostream>
#include <fstream>
#include <sstream>
#include <math.h>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <cstdlib>                  // For atoi()
#include <assert.h>

#include "grabber.h"

//##########  Define New Image Types & Inline Functions  ###############
// Create a type to hold 2 pixels in packed YUV 422 packed format
typedef unsigned int YUYV32;

// Define functions for unpacking YUYV 422 format
#define Y0(n) (n & 0x000000ff)
#define CB(n) ((n >>  8) & 0x000000ff)
#define Y1(n) ((n >> 16) & 0x000000ff)
#define CR(n) ((n >> 24) & 0x000000ff)

// Define function for packing a pixel into YUYV 422 format
#define YUYV(y0,u,y1,v) ((v<<24)+(y1<<16)+(u<<8)+(y0))

// Create a type to hold and RGB pixel
typedef unsigned int RGB32;

// Define functions for unpacking RGB32 pixel
#define RED(n)   ((n >> 16) & 0x000000FF)
#define GREEN(n) ((n >> 8) & 0x000000ff)
#define BLUE(n)  ((n) & 0x000000ff)

// Define function for packing a RGB32 pixel
#define RGB(r,g,b) ((0<<24) + (r<<16) + (g <<8) + (b))

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@  Set Resolution Here  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@  Only one of these can be uncommented at a time. The     @@@@@@
//@@@@  desired resolution also needs to be set in Grabber.cpp  @@@@@@

//#####################################
// Full NTSC resolution
#define HIGHRES 1
int Height = NTSC_HEIGHT;
int Width  = NTSC_WIDTH;

#define CropLeft 0
#define CropRight 0

// Dimensions of cropped images
int NewHeight = NTSC_HEIGHT;
int NewWidth = NTSC_WIDTH - CropLeft - CropRight;
```

```
YUYV32 Frame0[NTSC_HEIGHT*NTSC_WIDTH/2];
YUYV32 Frame1[NTSC_HEIGHT*NTSC_WIDTH/2];
YUYV32 Frame2[NTSC_HEIGHT*NTSC_WIDTH/2];
YUYV32 Frame3[NTSC_HEIGHT*NTSC_WIDTH/2];
//#### End Full NTSC resolution #######

// //########################################
// // 1/4 NTSC Resolution
// #define HIGHRES 0
// int Height = NTSC_HEIGHT/2;
// int Width  = NTSC_WIDTH/2;
//
// //   !!!!! Need to be EVEN numbers  !!!!!!!
// #define CropLeft 10
// #define CropRight 4
//
// // Dimensions of cropped images
// int NewHeight = NTSC_HEIGHT/2;
// int NewWidth  = NTSC_WIDTH/2 - CropLeft - CropRight;
//
// YUYV32 Frame0[NTSC_HEIGHT*NTSC_WIDTH/8];
// //#### End 1/4 NTSC Resolution ###########


int main(int argc, char *argv[])
{

  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  //@@@@@@@@@@ Variables  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

    int i, j;             // Counter variables
    int Row;
    char *name;           // Name to store picture files as ppm images
    char ans[10];         // Input from user

    int Yval = 0;         // Actual pixel values in YCbCr (YUV) space
    int Cbval = 0;
    int Crval = 0;

    YUYV32  pixels;       // Structure to hold packed YUYV 422 pixels (2 pixels)

    // Instantiate Grabber objects to grab frames
    Grabber Grab0;              // For camera 0
    Grabber Grab1;              // For camera 1
    Grabber Grab2;              // For camera 2
    Grabber Grab3;              // For camera 3

    std::ofstream mycolorfile;          // File to store color data for anal in Matlab
    std::ofstream grayfile;

    //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    //@@@@ End Variables  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


    //#################################################################################
    //############   Calibration Program   ##########################################
    //#################################################################################

    // Setup the Frame Grabber and Cameras
    // Open video device with YUV 422 format
    Grab0.Init(CF_422, "/dev/video0", true);
    Grab1.Init(CF_422, "/dev/video1", true);
    Grab2.Init(CF_422, "/dev/video2", true);
    Grab3.Init(CF_422, "/dev/video3", true);

    // Set channel of multiplexor for input
    Grab0.SetChannel("Composite0");
    Grab1.SetChannel("Composite0");
    Grab2.SetChannel("Composite0");
    Grab3.SetChannel("Composite0");

    // Video signal configure
    Grab0.SetVideoSignal("ntsc");
    Grab1.SetVideoSignal("ntsc");
    Grab2.SetVideoSignal("ntsc");
      Grab3.SetVideoSignal("ntsc");

    Grab0.SetByteOrder(BYTE_ORDER_YUYV);
    Grab1.SetByteOrder(BYTE_ORDER_YUYV);
    Grab2.SetByteOrder(BYTE_ORDER_YUYV);
    Grab3.SetByteOrder(BYTE_ORDER_YUYV);

    // Start grabbing
    Grab0.Start();
    Grab1.Start();
    Grab2.Start();
    Grab3.Start();

    // Inform User of Calibration Process
```

```
std::cout << "\n\nWe will capture four images, one from each camera, of\n";
std::cout << "the calibration print. \n\n";

//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&     Camera 0     &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

std::cout << "\n%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
std::cout << "\tFirst, we will measure Camera 0\n";
std::cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";

std::cout << "Place the calibration print under camera 0. Use the zoom\n";
std::cout << "feature on the camera to fill the camera frame with the\n";
std::cout << "calibration print. Be sure to get all of the colors in the\n";
std::cout << "view. Also make sure the camera is in focus. If not, it will\n";
std::cout << "act like a low pass filter on the image.\n\n";

std::cout << "Press 'y' then 'Enter' when you are ready:\t";
std::cin >> ans;
std::cout << "\n";

// Get frame from camera 1
Grab0.Grab();

// Quickly copy frame into array Frame0 for processing
Grab0.CopyFrame((unsigned char *)&Frame0[0]);

// Open File Stream and write header for PPM image
mycolorfile.open("Frames/Cam0.ppm", std::ios::out);

// Write Header for file
mycolorfile << "P3" << std::endl;
mycolorfile << NewWidth << " " << NewHeight << "\n";
mycolorfile << "255" << std::endl;

// Open File Stream and write header for PPM image
grayfile.open("Frames/grayCam0.pgm", std::ios::out);

// Write Header for file
grayfile << "P2" << std::endl;
grayfile << NewWidth << " " << NewHeight << "\n";
grayfile << "255" << std::endl;

// Move through the image  while cropping the right and left
//   "artifacts" created by the frame grabber
for(i = 0; i < NewHeight; i++)
{
  Row = i*Width/2;

  for(j = 0; j < (NewWidth)/2; j++)
  {

    // Get 2 packed pixels from array
    pixels = Frame0[Row+j+CropLeft/2];

    // Extract first pixel from packed pixel
    Yval  = Y0(pixels);
    Cbval = CB(pixels);
    Crval = CR(pixels);

    // Store Converted pixels to analyze in Matlab
    mycolorfile << Yval << " " << Cbval << " " << Crval << " ";
    grayfile << Yval << " ";

    // Extract second pixel from packed pixel
    Yval  = Y1(pixels);
    Cbval = CB(pixels);
    Crval = CR(pixels);

    // Store Converted pixels to analyze in Matlab
    mycolorfile << Yval << " " << Cbval << " " << Crval << " ";
    grayfile << Yval << " ";

    // Note: We are saving this knowing that it will be interpreted
    //  as an RGB image. So, a program reading this will display
    //  some interesting results, however, we are using MATLab to
    //  later analyze the image and the mfile knows these values
    //  are actually YCbCr instead of RGB

  }

}

// Close Color File Stream
mycolorfile.close();
grayfile.close();

usleep(10000000);

//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&     Camera 1     &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
```

```
std::cout << "\n##########################################################\n";
std::cout << "\tNow we will measure Camera 1\n";
std::cout << "##########################################################\n\n";

std::cout << "Place the calibration print under camera 1. Use the zoom\n";
std::cout << "feature on the camera to fill the camera frame with the\n";
std::cout << "calibration print. Be sure to get all of the colors in the\n";
std::cout << "view. Also make sure the camera is in focus. If not, it will\n";
std::cout << "act like a low pass filter on the image.\n\n";

std::cout << "Press 'y' then 'Enter' when you are ready:\t";
std::cin >> ans;
std::cout << "\n";

// Get frame from camera 1
Grab1.Grab();

// Quickly copy frame into array Frame0 for processing
Grab1.CopyFrame((unsigned char *)&Frame1[0]);

// Open File Stream and write header for PPM image
mycolorfile.open("Frames/Cam1.ppm", std::ios::out);

// Write Header for file
mycolorfile << "P3" << std::endl;
mycolorfile << NewWidth << " " << NewHeight << "\n";
mycolorfile << "255" << std::endl;

// Open File Stream and write header for PPM image
grayfile.open("Frames/grayCam1.pgm", std::ios::out);

// Write Header for file
grayfile << "P2" << std::endl;
grayfile << NewWidth << " " << NewHeight << "\n";
grayfile << "255" << std::endl;

// Move through the image  while cropping the right and left
//   "artifacts" created by the frame grabber
for(i = 0; i < NewHeight; i++)
{
  Row = i*Width/2;

  for(j = 0; j < (NewWidth)/2; j++)
  {

    // Get 2 packed pixels from array
    pixels = Frame0[Row+j+CropLeft/2];

    // Extract first pixel from packed pixel
    Yval  = Y0(pixels);
    Cbval = CB(pixels);
    Crval = CR(pixels);

    // Store Converted pixels to analyze in Matlab
    mycolorfile << Yval << " " << Cbval << " " << Crval << " ";
    grayfile << Yval << " ";

    // Extract second pixel from packed pixel
    Yval  = Y1(pixels);
    Cbval = CB(pixels);
    Crval = CR(pixels);

    // Store Converted pixels to analyze in Matlab
    mycolorfile << Yval << " " << Cbval << " " << Crval << " ";
    grayfile << Yval << " ";

    // Note: We are saving this knowing that it will be interpreted
    //  as an RGB image. So, a program reading this will display
    //  some interesting results, however, we are using MATLab to
    //  later analyze the image and the mfile knows these values
    //  are actually YCbCr instead of RGB

  }

}

// Close Color File Stream
mycolorfile.close();
grayfile.close();

usleep(10000000);

//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&  Camera 2  &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

std::cout << "\n##########################################################\n";
std::cout << "\tNow we will measure Camera 2\n";
std::cout << "##########################################################\n\n";

std::cout << "Place the calibration print under camera 2. Use the zoom\n";
```

```
std::cout << "feature on the camera to fill the camera frame with the\n";
std::cout << "calibration print. Be sure to get all of the colors in the\n";
std::cout << "view. Also make sure the camera is in focus. If not, it will\n";
std::cout << "act like a low pass filter on the image.\n\n";

std::cout << "Press 'y' then 'Enter' when you are ready:\t";
std::cin >> ans;
std::cout << "\n";

// Get frame from camera 2
Grab2.Grab();

// Quickly copy frame into array Frame0 for processing
Grab2.CopyFrame((unsigned char *)&Frame2[0]);

// Open File Stream and write header for PPM image
mycolorfile.open("Frames/Cam2.ppm", std::ios::out);

// Write Header for file
mycolorfile << "P3" << std::endl;
mycolorfile << NewWidth << " " << NewHeight << "\n";
mycolorfile << "255" << std::endl;

// Open File Stream and write header for PPM image
grayfile.open("Frames/grayCam2.pgm", std::ios::out);

// Write Header for file
grayfile << "P2" << std::endl;
grayfile << NewWidth << " " << NewHeight << "\n";
grayfile << "255" << std::endl;

// Move through the image  while cropping the right and left
//   "artifacts" created by the frame grabber
for(i = 0; i < NewHeight; i++)
{
  Row = i*Width/2;

  for(j = 0; j < (NewWidth)/2; j++)
  {

    // Get 2 packed pixels from array
    pixels = Frame0[Row+j+CropLeft/2];

    // Extract first pixel from packed pixel
    Yval  = Y0(pixels);
    Cbval = CB(pixels);
    Crval = CR(pixels);

    // Store Converted pixels to analyze in Matlab
    mycolorfile << Yval << " " << Cbval << " " << Crval << " ";
    grayfile << Yval << " ";

    // Extract second pixel from packed pixel
    Yval  = Y1(pixels);
    Cbval = CB(pixels);
    Crval = CR(pixels);

    // Store Converted pixels to analyze in Matlab
    mycolorfile << Yval << " " << Cbval << " " << Crval << " ";
    grayfile << Yval << " ";

    // Note: We are saving this knowing that it will be interpreted
    //  as an RGB image. So, a program reading this will display
    //  some interesting results, however, we are using MATLab to
    //  later analyze the image and the mfile knows these values
    //  are actually YCbCr instead of RGB

  }

}

// Close Color File Stream
mycolorfile.close();
grayfile.close();

usleep(10000000);

//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&    Camera 3    &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

std::cout << "\n%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n";
std::cout << "\tNow we will measure Camera 3\n";
std::cout << "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n\n";

std::cout << "Place the calibration print under camera 3. Use the zoom\n";
std::cout << "feature on the camera to fill the camera frame with the\n";
std::cout << "calibration print. Be sure to get all of the colors in the\n";
std::cout << "view. Also make sure the camera is in focus. If not, it will\n";
std::cout << "act like a low pass filter on the image.\n\n";

std::cout << "Press 'y' then 'Enter' when you are ready:\t";
```

```
    std::cin >> ans;
    std::cout << "\n";

    // Get frame from camera 3
    Grab3.Grab();

    // Quickly copy frame into array Frame0 for processing
    Grab3.CopyFrame((unsigned char *)&Frame3[0]);

    // Open File Stream and write header for PPM image
    mycolorfile.open("Frames/Cam3.ppm", std::ios::out);

    // Write Header for file
    mycolorfile << "P3" << std::endl;
    mycolorfile << NewWidth << " " << NewHeight << "\n";
    mycolorfile << "255" << std::endl;

    // Open File Stream and write header for PPM image
    grayfile.open("Frames/grayCam3.pgm", std::ios::out);

    // Write Header for file
    grayfile << "P2" << std::endl;
    grayfile << NewWidth << " " << NewHeight << "\n";
    grayfile << "255" << std::endl;

    // Move through the image  while cropping the right and left
    //   "artifacts" created by the frame grabber
    for(i = 0; i < NewHeight; i++)
    {
      Row = i*Width/2;

      for(j = 0; j < (NewWidth)/2; j++)
      {

        // Get 2 packed pixels from array
        pixels = Frame0[Row+j+CropLeft/2];

        // Extract first pixel from packed pixel
        Yval  = Y0(pixels);
        Cbval = CB(pixels);
        Crval = CR(pixels);

        // Store Converted pixels to analyze in Matlab
        mycolorfile << Yval << " " << Cbval << " " << Crval << " ";
        grayfile << Yval << " ";

        // Extract second pixel from packed pixel
        Yval  = Y1(pixels);
        Cbval = CB(pixels);
        Crval = CR(pixels);

        // Store Converted pixels to analyze in Matlab
        mycolorfile << Yval << " " << Cbval << " " << Crval << " ";
        grayfile << Yval << " ";

        // Note: We are saving this knowing that it will be interpreted
        //  as an RGB image. So, a program reading this will display
        //  some interesting results, however, we are using MATLab to
        //  later analyze the image and the mfile knows these values
        //  are actually YCbCr instead of RGB

      }

    }

    // Close Color File Stream
    mycolorfile.close();
    grayfile.close();

    usleep(10000000);

    // Stop Grabbing Frames
    Grab0.Stop();
    Grab1.Stop();
    Grab2.Stop();
    Grab3.Stop();


    // Done with Color Measurements and Calculations
    std::cout << "Color Calibration is Complete\n";

    return 0;
}
```

APPENDIX V

Color Threshold Calibration Program

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%        Calibrate Color Thresholds
%            by Travis Riggs
%
%    This program will take 4 images grabbed
%    from the testbed of the color calibration
%    pattern. It will calculate the statistics
%    of each color for each camera and use that
%    to set the global threshold levels.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Start fresh
clear all;
close all;
clc;

for Cameras = 1:4

    switch Cameras

        case 1
            prefix = 'Data/Cam0/';
            cam = 'Cam0';

        case 2
            prefix = 'Data/Cam1/';
            cam = 'Cam1';

        case 3
            prefix = 'Data/Cam2/';
            cam = 'Cam2';

        case 4
            prefix = 'Data/Cam3/';
            cam = 'Cam3';

        otherwise
            prefix = '';
    end

    close all;

    % Immport PGM file of YCbCr test pattern image
    pic = imread([prefix cam '.ppm']);

    % Convert it to RGB for display purposes
    pic = im2double(pic);
    displaypic = ycbcr2rgb(pic);

    % Display an image of test pattern
    DispHandle = figure;
    imshow(displaypic);

    pic = 255*pic;

    [rows, cols, dep] = size(pic);

    for Colors = 1:5

        switch Colors

            case 1
                colorName = 'Red';

            case 2
                colorName = 'Green';

            case 3
                colorName = 'Blue';

            case 4
                colorName = 'Black';
```

```
    case 5
        colorName = 'White';

    otherwise
        disp('ERROR in switch 1')
end

% Get 2 points from image
figure(DispHandle);

title(['Click the UPPER LEFT corner of the ' colorName ' square']);

[x1 y1] = ginput(1);

figure(DispHandle);

title(['Now, click the LOWER RIGHT corner of the ' colorName ' square']);

[x2 y2] = ginput(1);

X = [x1 x2];
Y = [y1 y2];

X = round(X);
Y = round(Y);

disp(['Got the points .... now calculating statistics for ' colorName]);

%%%% DEBUG
figure, imshow(displaypic(Y(1):Y(2),X(1):X(2),:));


%% Calculate color statistics for thresholds
k = 1;
for i = Y(1):Y(2)
    for j = X(1):X(2)

        % Extract channel values into vector
        tempY(k)  = pic(i,j,1);
        tempCb(k) = pic(i,j,2);
        tempCr(k) = pic(i,j,3);

        k = k + 1;

    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate statistics

% Find mins
tempYmin  = min(tempY);
tempCbmin = min(tempCb);
tempCrmin = min(tempCr);

% Find maxs
tempYmax  = max(tempY);
tempCbmax = max(tempCb);
tempCrmax = max(tempCr);

% Find means
tempYmean  = mean(tempY);
tempCbmean = mean(tempCb);
tempCrmean = mean(tempCr);

% Find Stdeviation
tempYstd  = std(tempY);
tempCbstd = std(tempCb);
tempCrstd = std(tempCr);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Find Normalized Histogram
nhist = zeros(1,256);

for i = 1:length(tempY);
    nhist(tempY(i)+1) =  nhist(tempY(i)+1) + 1;
end

Yhist = nhist/(length(tempY));          % Normalize histogram

nhist = zeros(1,256);

for i = 1:length(tempCb)
    nhist(tempCb(i)+1) =  nhist(tempCb(i)+1) + 1;
end

Cbhist = nhist/(length(tempCb));          % Normalize histogram

nhist = zeros(1,256);

for i = 1:length(tempCr)
    nhist(tempCr(i)+1) =  nhist(tempCr(i)+1) + 1;
```

```
        end

        Crhist = nhist/(length(tempCb));          % Normalize histogram

        clear nhist;


        %%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %    Plot histograms
        MYHIST = figure;
        subplot(3,1,1)
        stem(0:255, Yhist, 'k.');
        title([cam ': The Color ' colorName ' Normalized YCbCr Histograms']);
        ylabel('Y');
        axis([0 255 0 1.2*max(Yhist)]);

        subplot(3,1,2)
        stem(0:255, Cbhist, 'b.');
        ylabel('Cb');
        axis([0 255 0 1.2*max(Cbhist)]);

        subplot(3,1,3)
        stem(0:255, Crhist, 'r.');
        ylabel('Cr');
        axis([0 255 0 1.2*max(Crhist)]);

        saveas(MYHIST, ['Plots/' cam colorName 'YUVHist.tif']);

        % Write Pixel Data to txt file
        myData = [tempY' tempCb' tempCr'];

        fid = fopen([prefix colorName 'Data.txt'],'w');
        fprintf(fid,'%3.0f\t %3.0f\t %3.0f\n', myData);
        fclose(fid);

        %% Write pixel data to 'Total' file
        % if(Cameras == 1)
        %     fid = fopen(['Data/Total' colorName '.txt'], 'w');
        %else
        %     fid = fopen(['Data/Total' colorName '.txt'], 'a');
        %end
        %fprintf(fid,'%3.0f\t %3.0f\t %3.0f\n', myData);
        %fclose(fid);


        % Write Statistics Data to ColorStats.txt
        if Colors == 1
            fid = fopen([prefix 'ColorStats.txt'],'w');
        else
            fid = fopen([prefix 'ColorStats.txt'],'a');
        end

        fprintf(fid,'%s:\n', colorName);

        line = sprintf('YMin=%d,YMean=%.3f,YStd=%.3f,YMax=%d',tempYmin,tempYmean,tempYstd,tempYmax);
        fprintf(fid,'%s\n', line);

        line = sprintf('CbMin=%d,CbMean=%.3f,CbStd=%.3f,CbMax=%d',tempCbmin,tempCbmean,tempCbstd,tempCbmax);
        fprintf(fid,'%s\n', line);

        line = sprintf('CrMin=%d,CrMean=%.3f,CrStd=%.3f,CrMax=%d',tempCrmin,tempCrmean,tempCrstd,tempCrmax);
        fprintf(fid,'%s\n\n', line);

        fclose(fid);

        % Clear variables for next color
        clear tempY tempCr tempCb myData

    end

    % Clear for next camera
    clear pic displaypic

end

disp('Individual Camera calculations completed. Run ');
disp('TotalColorCalibration for the total statistics');
```

APPENDIX VI

Kernel Modification Files

```
/*
 * USB FTDI SIO driver
 *
 *      Copyright (C) 1999 - 2001
 *          Greg Kroah-Hartman (greg@kroah.com)
 *          Bill Ryder (bryder@sgi.com)
 *      Copyright (C) 2002
 *          Kuba Ober (kuba@mareimbrium.org)
 *
 *      This program is free software; you can redistribute it and/or modify
 *      it under the terms of the GNU General Public License as published by
 *      the Free Software Foundation; either version 2 of the License, or
 *      (at your option) any later version.
 *
 * See Documentation/usb/usb-serial.txt for more information on using this driver
 *
 * See http://ftdi-usb-sio.sourceforge.net for upto date testing info
 *      and extra documentation
 *
 * (21/Jul/2004) Ian Abbott
 *      Incorporated Steven Turner's code to add support for the FT2232C chip.
 *      The prelimilary port to the 2.6 kernel was by Rus V. Brushkoff.  I have
 *      fixed a couple of things.
 *
 * (27/May/2004) Ian Abbott
 *      Improved throttling code, mostly stolen from the WhiteHEAT driver.
 *
 * (26/Mar/2004) Jan Capek
 *      Added PID's for ICD-U20/ICD-U40 - incircuit PIC debuggers from CCS Inc.
 *
 * (09/Feb/2004) Ian Abbott
 *      Changed full name of USB-UIRT device to avoid "/" character.
 *      Added FTDI's alternate PID (0x6006) for FT232/245 devices.
 *      Added PID for "ELV USB Module UO100" from Stefan Frings.
 *
 * (21/Oct/2003) Ian Abbott
 *      Renamed some VID/PID macros for Matrix Orbital and Perle Systems
 *      devices.  Removed Matrix Orbital and Perle Systems devices from the
 *      8U232AM device table, but left them in the FT232BM table, as they are
 *      known to use only FT232BM.
 *
 * (17/Oct/2003) Scott Allen
 *      Added vid/pid for Perle Systems UltraPort USB serial converters
 *
 * (21/Sep/2003) Ian Abbott
 *      Added VID/PID for Omnidirectional Control Technology US101 USB to
 *      RS-232 adapter (also rebadged as Dick Smith Electronics XH6381).
 *      VID/PID supplied by Donald Gordon.
 *
 * (19/Aug/2003) Ian Abbott
 *      Freed urb's transfer buffer in write bulk callback.
 *      Omitted some paranoid checks in write bulk callback that don't matter.
 *      Scheduled work in write bulk callback regardless of port's open count.
 *
 * (05/Aug/2003) Ian Abbott
 *      Added VID/PID for ID TECH IDT1221U USB to RS-232 adapter.
 *      VID/PID provided by Steve Briggs.
 *
 * (23/Jul/2003) Ian Abbott
 *      Added PIDs for CrystalFontz 547, 633, 631, 635, 640 and 640 from
 *      Wayne Wylupski.
 *
 * (10/Jul/2003) David Glance
 *      Added PID for DSS-20 SyncStation cradle for Sony-Ericsson P800.
 *
 * (27/Jun/2003) Ian Abbott
 *      Reworked the urb handling logic.  We have no more pool, but dynamically
 *      allocate the urb and the transfer buffer on the fly.  In testing this
 *      does not incure any measurable overhead.  This also relies on the fact
 *      that we have proper reference counting logic for urbs.  I nicked this
 *      from Greg KH's Visor driver.
 *
 * (23/Jun/2003) Ian Abbott
 *      Reduced flip buffer pushes and corrected a data length test in
 *      ftdi_read_bulk_callback.
```

```
*      Defererence pointers after any paranoid checks, not before.
*
* (21/Jun/2003) Erik Nygren
*      Added support for Home Electronics Tira-1 IR transceiver using FT232BM chip.
*      See <http://www.home-electro.com/tira1.htm>.  Only operates properly
*      at 100000 and RTS-CTS, so set custom divisor mode on startup.
*      Also force the Tira-1 and USB-UIRT to only use their custom baud rates.
*
* (18/Jun/2003) Ian Abbott
*      Added Device ID of the USB relais from Rudolf Gugler (backported from
*      Philipp Ghring's patch for 2.5.x kernel).
*      Moved read transfer buffer reallocation into startup function.
*      Free existing write urb and transfer buffer in startup function.
*      Only use urbs in write urb pool that were successfully allocated.
*      Moved some constant macros out of functions.
*      Minor whitespace and comment changes.
*
* (12/Jun/2003) David Norwood
*      Added support for USB-UIRT IR transceiver using 8U232AM chip.
*      See <http://home.earthlink.net/~jrhees/USBUIRT/index.htm>.  Only
*      operates properly at 312500, so set custom divisor mode on startup.
*
* (12/Jun/2003) Ian Abbott
*      Added Sealevel SeaLINK+ 210x, 220x, 240x, 280x vid/pids from Tuan Hoang
*      - I've eliminated some that don't seem to exist!
*      Added Home Electronics Tira-1 IR transceiver pid from Chris Horn
*      Some whitespace/coding-style cleanups
*
* (11/Jun/2003) Ian Abbott
*      Fixed unsafe spinlock usage in ftdi_write
*
* (24/Feb/2003) Richard Shooter
*      Increase read buffer size to improve read speeds at higher baud rates
*      (specifically tested with up to 1Mb/sec at 1.5M baud)
*
* (23/Feb/2003) John Wilkins
*      Added Xon/xoff flow control (activating support in the ftdi device)
*      Added vid/pid for Videonetworks/Homechoice (UK ISP)
*
* (23/Feb/2003) Bill Ryder
*      Added matrix orb device vid/pids from Wayne Wylupski
*
* (19/Feb/2003) Ian Abbott
*      For TIOCSSERIAL, set alt_speed to 0 when ASYNC_SPD_MASK value has
*      changed to something other than ASYNC_SPD_HI, ASYNC_SPD_VHI,
*      ASYNC_SPD_SHI or ASYNC_SPD_WARP.  Also, unless ASYNC_SPD_CUST is in
*      force, don't bother changing baud rate when custom_divisor has changed.
*
* (18/Feb/2003) Ian Abbott
*      Fixed TIOCMGET handling to include state of DTR and RTS, the state
*      of which are now saved by set_dtr() and set_rts().
*      Fixed improper storage class for buf in set_dtr() and set_rts().
*      Added FT232BM chip type and support for its extra baud rates (compared
*      to FT8U232AM).
*      Took account of special case divisor values for highest baud rates of
*      FT8U232AM and FT232BM.
*      For TIOCSSERIAL, forced alt_speed to 0 when ASYNC_SPD_CUST kludge used,
*      as previous alt_speed setting is now stale.
*      Moved startup code common between the startup routines for the
*      different chip types into a common subroutine.
*
* (17/Feb/2003) Bill Ryder
*      Added write urb buffer pool on a per device basis
*      Added more checking for open file on callbacks (fixed OOPS)
*      Added CrystalFontz 632 and 634 PIDs
*         (thanx to CrystalFontz for the sample devices - they flushed out
*           some driver bugs)
*      Minor debugging message changes
*      Added throttle, unthrottle and chars_in_buffer functions
*      Fixed FTDI_SIO (the original device) bug
*      Fixed some shutdown handling
*
*
*
*
* (07/Jun/2002) Kuba Ober
*      Changed FTDI_SIO_BASE_BAUD_TO_DIVISOR macro into ftdi_baud_to_divisor
*      function. It was getting too complex.
*      Fix the divisor calculation logic which was setting divisor of 0.125
*      instead of 0.5 for fractional parts of divisor equal to 5/8, 6/8, 7/8.
*      Also make it bump up the divisor to next integer in case of 7/8 - it's
*      a better approximation.
*
* (25/Jul/2002) Bill Ryder inserted Dmitri's TIOCMIWAIT patch
*      Not tested by me but it doesn't break anything I use.
*
* (04/Jan/2002) Kuba Ober
*      Implemented 38400 baudrate kludge, where it can be substituted with other
*        values. That's the only way to set custom baudrates.
*      Implemented TIOCSSERIAL, TIOCGSERIAL ioctl's so that setserial is happy.
*      FIXME: both baudrate things should eventually go to usbserial.c as other
*        devices may need that functionality too. Actually, it can probably be
```

```
 *        merged in serial.c somehow - too many drivers repeat this code over
 *        and over.
 *      Fixed baudrate forgetfulness - open() used to reset baudrate to 9600 every time.
 *      Divisors for baudrates are calculated by a macro.
 *      Small code cleanups. Ugly whitespace changes for Plato's sake only ;-].
 *
 * (04/Nov/2001) Bill Ryder
 *      Fixed bug in read_bulk_callback where incorrect urb buffer was used.
 *      Cleaned up write offset calculation
 *      Added write_room since default values can be incorrect for sio
 *      Changed write_bulk_callback to use same queue_task as other drivers
 *        (the previous version caused panics)
 *      Removed port iteration code since the device only has one I/O port and it
 *        was wrong anyway.
 *
 * (31/May/2001) gkh
 *      Switched from using spinlock to a semaphore, which fixes lots of problems.
 *
 * (23/May/2001)   Bill Ryder
 *      Added runtime debug patch (thanx Tyson D Sawyer).
 *      Cleaned up comments for 8U232
 *      Added parity, framing and overrun error handling
 *      Added receive break handling.
 *
 * (04/08/2001) gb
 *      Identify version on module load.
 *
 * (18/March/2001) Bill Ryder
 *      (Not released)
 *      Added send break handling. (requires kernel patch too)
 *      Fixed 8U232AM hardware RTS/CTS etc status reporting.
 *      Added flipbuf fix copied from generic device
 *
 * (12/3/2000) Bill Ryder
 *      Added support for 8U232AM device.
 *      Moved PID and VIDs into header file only.
 *      Turned on low-latency for the tty (device will do high baudrates)
 *      Added shutdown routine to close files when device removed.
 *      More debug and error message cleanups.
 *
 * (11/13/2000) Bill Ryder
 *      Added spinlock protected open code and close code.
 *      Multiple opens work (sort of - see webpage mentioned above).
 *      Cleaned up comments. Removed multiple PID/VID definitions.
 *      Factorised cts/dtr code
 *      Made use of __FUNCTION__ in dbg's
 *
 * (11/01/2000) Adam J. Richter
 *      usb_device_id table support
 *
 * (10/05/2000) gkh
 *      Fixed bug with urb->dev not being set properly, now that the usb
 *      core needs it.
 *
 * (09/11/2000) gkh
 *      Removed DEBUG #ifdefs with call to usb_serial_debug_data
 *
 * (07/19/2000) gkh
 *      Added module_init and module_exit functions to handle the fact that this
 *      driver is a loadable module now.
 *
 * (04/04/2000) Bill Ryder
 *      Fixed bugs in TCGET/TCSET ioctls (by removing them - they are
 *        handled elsewhere in the tty io driver chain).
 *
 * (03/30/2000) Bill Ryder
 *      Implemented lots of ioctls
 *      Fixed a race condition in write
 *      Changed some dbg's to errs
 *
 * (03/26/2000) gkh
 *      Split driver up into device specific pieces.
 *
 */

/* Bill Ryder - bryder@sgi.com - wrote the FTDI_SIO implementation */
/* Thanx to FTDI for so kindly providing details of the protocol required */
/*   to talk to the device */
/* Thanx to gkh and the rest of the usb dev group for all code I have assimilated :-) */

#include <linux/config.h>
#include <linux/kernel.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/tty.h>
#include <linux/tty_driver.h>
#include <linux/tty_flip.h>
#include <linux/module.h>
#include <linux/spinlock.h>
#include <asm/uaccess.h>
#include <linux/usb.h>
```

```
#include <linux/serial.h>
#include "usb-serial.h"
#include "ftdi_sio.h"

/*
 * Version Information
 */
#define DRIVER_VERSION "v1.4.2"
#define DRIVER_AUTHOR "Greg Kroah-Hartman <greg@kroah.com>, Bill Ryder <bryder@sgi.com>, Kuba Ober <kuba@mareimbrium.org>"
#define DRIVER_DESC "USB FTDI Serial Converters Driver"

static int debug;

static struct usb_device_id id_table_sio [] = {
        { USB_DEVICE(FTDI_VID, FTDI_SIO_PID) },
        { USB_DEVICE(MOBILITY_VID, MOBILITY_USB_SERIAL_PID) },
        { }                                             /* Terminating entry */
};

/*
 * The 8U232AM has the same API as the sio except for:
 * - it can support MUCH higher baudrates; up to:
 *   o 921600 for RS232 and 2000000 for RS422/485 at 48MHz
 *   o 230400 at 12MHz
 *   so .. 8U232AM's baudrate setting codes are different
 * - it has a two byte status code.
 * - it returns characters every 16ms (the FTDI does it every 40ms)
 *
 * the bcdDevice value is used to differentiate FT232BM and FT245BM from
 * the earlier FT8U232AM and FT8U232BM.  For now, include all known VID/PID
 * combinations in both tables.
 * FIXME: perhaps bcdDevice can also identify 12MHz devices, but I don't know
 * if those ever went into mass production. [Ian Abbott]
 */


static struct usb_device_id id_table_8U232AM [] = {
        { USB_DEVICE_VER(FTDI_VID, FTDI_IRTRANS_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_8U232AM_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_8U232AM_ALT_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_RELAIS_PID, 0, 0x3ff) },
        { USB_DEVICE(INTERBIOMETRICS_VID, INTERBIOMETRICS_IOBOARD_PID) },
        { USB_DEVICE(INTERBIOMETRICS_VID, INTERBIOMETRICS_MINI_IOBOARD_PID) },
        { USB_DEVICE_VER(FTDI_NF_RIC_VID, FTDI_NF_RIC_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_632_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_634_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_547_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_633_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_631_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_635_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_640_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_642_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_VNHCPCUSB_D_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_DSS20_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2101_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2102_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2103_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2104_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2201_1_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2201_2_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2202_1_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2202_2_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2203_1_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2203_2_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2401_1_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2401_2_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2401_3_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2401_4_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2402_1_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2402_2_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2402_3_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2402_4_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2403_1_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2403_2_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2403_3_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2403_4_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_1_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_2_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_3_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_4_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_5_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_6_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_7_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_8_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_1_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_2_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_3_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_4_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_5_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_6_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_7_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_8_PID, 0, 0x3ff) },
```

```
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_1_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_2_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_3_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_4_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_5_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_6_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_7_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_8_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(IDTECH_VID, IDTECH_IDT1221U_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(OCT_VID, OCT_US101_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, PROTEGO_SPECIAL_1, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, PROTEGO_R2X0, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, PROTEGO_SPECIAL_3, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, PROTEGO_SPECIAL_4, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_ELV_UO100_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_ELV_UM100_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, INSIDE_ACCESSO, 0, 0x3ff) },
        { USB_DEVICE_VER(INTREPID_VID, INTREPID_VALUECAN_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(INTREPID_VID, INTREPID_NEOVI_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FALCOM_VID, FALCOM_TWIST_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_SUUNTO_SPORTS_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_RM_CANVIEW_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(BANDB_VID, BANDB_USOTL4_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(BANDB_VID, BANDB_USTL4_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(BANDB_VID, BANDB_USO9ML2_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, EVER_ECO_PRO_CDS, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_4N_GALAXY_DE_0_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_4N_GALAXY_DE_1_PID, 0, 0x3ff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_4N_GALAXY_DE_2_PID, 0, 0x3ff) },
        { USB_DEVICE(BLACKCAT_VID, BLACKCAT_GM10_PID) },
        { USB_DEVICE(EVOLUTION_VID, EVOLUTION_ER1_PID) },
        { }                                             /* Terminating entry */
};

static struct usb_device_id id_table_FT232BM [] = {
        { USB_DEVICE_VER(FTDI_VID, FTDI_IRTRANS_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_8U232AM_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_8U232AM_ALT_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_RELAIS_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_NF_RIC_VID, FTDI_NF_RIC_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_632_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_634_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_547_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_633_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_631_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_635_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_640_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_XF_642_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_VNHCPCUSB_D_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_DSS20_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_0_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_3_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_4_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_5_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_6_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_PERLE_ULTRAPORT_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_PIEGROUP_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2101_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2102_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2103_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2104_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2201_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2201_2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2202_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2202_2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2203_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2203_2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2401_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2401_2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2401_3_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2401_4_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2402_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2402_2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2402_3_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2402_4_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2403_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2403_2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2403_3_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2403_4_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_3_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_4_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_5_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_6_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_7_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2801_8_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_2_PID, 0x400, 0xffff) },
```

```
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_3_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_4_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_5_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_6_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_7_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2802_8_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_3_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_4_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_5_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_6_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_7_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(SEALEVEL_VID, SEALEVEL_2803_8_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(IDTECH_VID, IDTECH_IDT1221U_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(OCT_VID, OCT_US101_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, PROTEGO_SPECIAL_1, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, PROTEGO_R2X0, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, PROTEGO_SPECIAL_3, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, PROTEGO_SPECIAL_4, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E808_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E809_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80A_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80B_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80C_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80D_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80E_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80F_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E888_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E889_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88A_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88B_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88C_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88D_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88E_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88F_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_ELV_UO100_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_ELV_UM100_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, LINX_SDMUSBQSS_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, LINX_MASTERDEVEL2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, LINX_FUTURE_0_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, LINX_FUTURE_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, LINX_FUTURE_2_PID, 0x400, 0xffff) },
        { USB_DEVICE(FTDI_VID, FTDI_CCSICDU20_0_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_CCSICDU40_1_PID) },
        { USB_DEVICE_VER(FTDI_VID, INSIDE_ACCESSO, 0x400, 0xffff) },
        { USB_DEVICE_VER(INTREPID_VID, INTREPID_VALUECAN_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(INTREPID_VID, INTREPID_NEOVI_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FALCOM_VID, FALCOM_TWIST_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_SUUNTO_SPORTS_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_RM_CANVIEW_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(BANDB_VID, BANDB_USOTL4_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(BANDB_VID, BANDB_USTL4_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(BANDB_VID, BANDB_USO9ML2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, EVER_ECO_PRO_CDS, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_4N_GALAXY_DE_0_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_4N_GALAXY_DE_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_4N_GALAXY_DE_2_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_ACTIVE_ROBOTS_PID, 0x400, 0xffff) },
        { }                                             /* Terminating entry */
};


static struct usb_device_id id_table_USB_UIRT [] = {
        { USB_DEVICE(FTDI_VID, FTDI_USB_UIRT_PID) },
        { }                                             /* Terminating entry */
};


static struct usb_device_id id_table_HE_TIRA1 [] = {
        { USB_DEVICE_VER(FTDI_VID, FTDI_HE_TIRA1_PID, 0x400, 0xffff) },
        { }                                             /* Terminating entry */
};


static struct usb_device_id id_table_FT2232C[] = {
        { USB_DEVICE(FTDI_VID, FTDI_8U2232C_PID) },
        { }                                             /* Terminating entry */
};


static struct usb_device_id id_table_combined [] = {
        { USB_DEVICE(FTDI_VID, FTDI_IRTRANS_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_SIO_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_8U232AM_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_8U232AM_ALT_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_8U2232C_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_RELAIS_PID) },
        { USB_DEVICE(INTERBIOMETRICS_VID, INTERBIOMETRICS_IOBOARD_PID) },
        { USB_DEVICE(INTERBIOMETRICS_VID, INTERBIOMETRICS_MINI_IOBOARD_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_XF_632_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_XF_634_PID) },
```

```
{ USB_DEVICE(FTDI_VID, FTDI_XF_547_PID) },
{ USB_DEVICE(FTDI_VID, FTDI_XF_633_PID) },
{ USB_DEVICE(FTDI_VID, FTDI_XF_631_PID) },
{ USB_DEVICE(FTDI_VID, FTDI_XF_635_PID) },
{ USB_DEVICE(FTDI_VID, FTDI_XF_640_PID) },
{ USB_DEVICE(FTDI_VID, FTDI_XF_642_PID) },
{ USB_DEVICE(FTDI_VID, FTDI_DSS20_PID) },
{ USB_DEVICE(FTDI_NF_RIC_VID, FTDI_NF_RIC_PID) },
{ USB_DEVICE(FTDI_VID, FTDI_VNHCPCUSB_D_PID) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_0_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_1_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_2_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_3_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_4_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_5_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_MTXORB_6_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_PERLE_ULTRAPORT_PID, 0x400, 0xffff) },
{ USB_DEVICE(FTDI_VID, FTDI_PIEGROUP_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2101_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2102_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2103_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2104_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2201_1_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2201_2_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2202_1_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2202_2_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2203_1_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2203_2_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2401_1_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2401_2_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2401_3_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2401_4_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2402_1_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2402_2_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2402_3_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2402_4_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2403_1_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2403_2_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2403_3_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2403_4_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2801_1_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2801_2_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2801_3_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2801_4_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2801_5_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2801_6_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2801_7_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2801_8_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2802_1_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2802_2_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2802_3_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2802_4_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2802_5_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2802_6_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2802_7_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2802_8_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2803_1_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2803_2_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2803_3_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2803_4_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2803_5_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2803_6_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2803_7_PID) },
{ USB_DEVICE(SEALEVEL_VID, SEALEVEL_2803_8_PID) },
{ USB_DEVICE(IDTECH_VID, IDTECH_IDT1221U_PID) },
{ USB_DEVICE(OCT_VID, OCT_US101_PID) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_HE_TIRA1_PID, 0x400, 0xffff) },
{ USB_DEVICE(FTDI_VID, FTDI_USB_UIRT_PID) },
{ USB_DEVICE(FTDI_VID, PROTEGO_SPECIAL_1) },
{ USB_DEVICE(FTDI_VID, PROTEGO_R2X0) },
{ USB_DEVICE(FTDI_VID, PROTEGO_SPECIAL_3) },
{ USB_DEVICE(FTDI_VID, PROTEGO_SPECIAL_4) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E808_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E809_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80A_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80B_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80C_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80D_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80E_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E80F_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E888_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E889_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88A_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88B_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88C_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88D_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88E_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, FTDI_GUDEADS_E88F_PID, 0x400, 0xffff) },
{ USB_DEVICE(FTDI_VID, FTDI_ELV_UO100_PID) },
{ USB_DEVICE(FTDI_VID, FTDI_ELV_UM100_PID) },
{ USB_DEVICE_VER(FTDI_VID, LINX_SDMUSBQSS_PID, 0x400, 0xffff) },
{ USB_DEVICE_VER(FTDI_VID, LINX_MASTERDEVEL2_PID, 0x400, 0xffff) },
```

```
        { USB_DEVICE_VER(FTDI_VID, LINX_FUTURE_0_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, LINX_FUTURE_1_PID, 0x400, 0xffff) },
        { USB_DEVICE_VER(FTDI_VID, LINX_FUTURE_2_PID, 0x400, 0xffff) },
        { USB_DEVICE(FTDI_VID, FTDI_CCSICDU20_0_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_CCSICDU40_1_PID) },
        { USB_DEVICE(FTDI_VID, INSIDE_ACCESSO) },
        { USB_DEVICE(INTREPID_VID, INTREPID_VALUECAN_PID) },
        { USB_DEVICE(INTREPID_VID, INTREPID_NEOVI_PID) },
        { USB_DEVICE(FALCOM_VID, FALCOM_TWIST_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_SUUNTO_SPORTS_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_RM_CANVIEW_PID) },
        { USB_DEVICE(BANDB_VID, BANDB_USOTL4_PID) },
        { USB_DEVICE(BANDB_VID, BANDB_USTL4_PID) },
        { USB_DEVICE(BANDB_VID, BANDB_USO9ML2_PID) },
        { USB_DEVICE(FTDI_VID, EVER_ECO_PRO_CDS) },
        { USB_DEVICE(FTDI_VID, FTDI_4N_GALAXY_DE_0_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_4N_GALAXY_DE_1_PID) },
        { USB_DEVICE(FTDI_VID, FTDI_4N_GALAXY_DE_2_PID) },
        { USB_DEVICE(MOBILITY_VID, MOBILITY_USB_SERIAL_PID) },
        { USB_DEVICE_VER(FTDI_VID, FTDI_ACTIVE_ROBOTS_PID, 0x400, 0xffff) },
        { USB_DEVICE(BLACKCAT_VID, BLACKCAT_GM10_PID) },
        { USB_DEVICE(EVOLUTION_VID, EVOLUTION_ER1_PID) },
        { }                                             /* Terminating entry */
};

MODULE_DEVICE_TABLE (usb, id_table_combined);

static struct usb_driver ftdi_driver = {
        .name =         "ftdi_sio",
        .probe =        usb_serial_probe,
        .disconnect =   usb_serial_disconnect,
        .id_table =     id_table_combined,
};

static char *ftdi_chip_name[] = {
        [SIO] = "SIO",  /* the serial part of FT8U100AX */
        [FT8U232AM] = "FT8U232AM",
        [FT232BM] = "FT232BM",
        [FT2232C] = "FT2232C",
};


/* Constants for read urb and write urb */
#define BUFSZ 512
#define PKTSZ 64

/* rx_flags */
#define THROTTLED               0x01
#define ACTUALLY_THROTTLED      0x02

struct ftdi_private {
        ftdi_chip_type_t chip_type;
                                /* type of the device, either SIO or FT8U232AM */
        int baud_base;          /* baud base clock for divisor setting */
        int custom_divisor;     /* custom_divisor kludge, this is for baud_base (different from what goes to the chip!) */
        __u16 last_set_data_urb_value ;
                                /* the last data state set - needed for doing a break */
        int write_offset;       /* This is the offset in the usb data block to write the serial data -
                                 * it is different between devices
                                 */
        int flags;              /* some ASYNC_xxxx flags are supported */
        unsigned long last_dtr_rts;     /* saved modem control outputs */
        wait_queue_head_t delta_msr_wait; /* Used for TIOCMIWAIT */
        char prev_status, diff_status;        /* Used for TIOCMIWAIT */
        __u8 rx_flags;          /* receive state flags (throttling) */
        spinlock_t rx_lock;     /* spinlock for receive state */
        struct work_struct rx_work;
        int rx_processed;

        __u16 interface;        /* FT2232C port interface (0 for FT232/245) */

        int force_baud;         /* if non-zero, force the baud rate to this value */
        int force_rtscts;       /* if non-zero, force RTS-CTS to always be enabled */
};

/* Used for TIOCMIWAIT */
#define FTDI_STATUS_B0_MASK     (FTDI_RS0_CTS | FTDI_RS0_DSR | FTDI_RS0_RI | FTDI_RS0_RLSD)
#define FTDI_STATUS_B1_MASK     (FTDI_RS_BI)
/* End TIOCMIWAIT */

#define FTDI_IMPL_ASYNC_FLAGS = ( ASYNC_SPD_HI | ASYNC_SPD_VHI \
 ASYNC_SPD_CUST | ASYNC_SPD_SHI | ASYNC_SPD_WARP )

/* function prototypes for a FTDI serial converter */
static int  ftdi_SIO_startup            (struct usb_serial *serial);
static int  ftdi_8U232AM_startup        (struct usb_serial *serial);
static int  ftdi_FT232BM_startup        (struct usb_serial *serial);
static int  ftdi_FT2232C_startup        (struct usb_serial *serial);
static int  ftdi_USB_UIRT_startup       (struct usb_serial *serial);
static int  ftdi_HE_TIRA1_startup       (struct usb_serial *serial);
static void ftdi_shutdown               (struct usb_serial *serial);
static int  ftdi_open                   (struct usb_serial_port *port, struct file *filp);
```

```
static void ftdi_close                  (struct usb_serial_port *port, struct file *filp);
static int  ftdi_write                  (struct usb_serial_port *port, const unsigned char *buf, int count);
static int  ftdi_write_room             (struct usb_serial_port *port);
static int  ftdi_chars_in_buffer        (struct usb_serial_port *port);
static void ftdi_write_bulk_callback    (struct urb *urb, struct pt_regs *regs);
static void ftdi_read_bulk_callback     (struct urb *urb, struct pt_regs *regs);
static void ftdi_process_read           (void *param);
static void ftdi_set_termios            (struct usb_serial_port *port, struct termios * old);
static int  ftdi_tiocmget               (struct usb_serial_port *port, struct file *file);
static int  ftdi_tiocmset               (struct usb_serial_port *port, struct file * file, unsigned int set, unsigned int clear);
static int  ftdi_ioctl                  (struct usb_serial_port *port, struct file * file, unsigned int cmd, unsigned long arg);
static void ftdi_break_ctl              (struct usb_serial_port *port, int break_state );
static void ftdi_throttle               (struct usb_serial_port *port);
static void ftdi_unthrottle             (struct usb_serial_port *port);

static unsigned short int ftdi_232am_baud_base_to_divisor (int baud, int base);
static unsigned short int ftdi_232am_baud_to_divisor (int baud);
static __u32 ftdi_232bm_baud_base_to_divisor (int baud, int base);
static __u32 ftdi_232bm_baud_to_divisor (int baud);

static struct usb_serial_device_type ftdi_SIO_device = {
        .owner =                THIS_MODULE,
        .name =                 "FTDI_SIO",
        .id_table =             id_table_sio,
        .num_interrupt_in =     0,
        .num_bulk_in =          1,
        .num_bulk_out =         1,
        .num_ports =            1,
        .open =                 ftdi_open,
        .close =                ftdi_close,
        .throttle =             ftdi_throttle,
        .unthrottle =           ftdi_unthrottle,
        .write =                ftdi_write,
        .write_room =           ftdi_write_room,
        .chars_in_buffer =      ftdi_chars_in_buffer,
        .read_bulk_callback =   ftdi_read_bulk_callback,
        .write_bulk_callback =  ftdi_write_bulk_callback,
        .tiocmget =             ftdi_tiocmget,
        .tiocmset =             ftdi_tiocmset,
        .ioctl =                ftdi_ioctl,
        .set_termios =          ftdi_set_termios,
        .break_ctl =            ftdi_break_ctl,
        .attach =               ftdi_SIO_startup,
        .shutdown =             ftdi_shutdown,
};

static struct usb_serial_device_type ftdi_8U232AM_device = {
        .owner =                THIS_MODULE,
        .name =                 "FTDI_8U232AM_Compatible",
        .id_table =             id_table_8U232AM,
        .num_interrupt_in =     0,
        .num_bulk_in =          1,
        .num_bulk_out =         1,
        .num_ports =            1,
        .open =                 ftdi_open,
        .close =                ftdi_close,
        .throttle =             ftdi_throttle,
        .unthrottle =           ftdi_unthrottle,
        .write =                ftdi_write,
        .write_room =           ftdi_write_room,
        .chars_in_buffer =      ftdi_chars_in_buffer,
        .read_bulk_callback =   ftdi_read_bulk_callback,
        .write_bulk_callback =  ftdi_write_bulk_callback,
        .tiocmget =             ftdi_tiocmget,
        .tiocmset =             ftdi_tiocmset,
        .ioctl =                ftdi_ioctl,
        .set_termios =          ftdi_set_termios,
        .break_ctl =            ftdi_break_ctl,
        .attach =               ftdi_8U232AM_startup,
        .shutdown =             ftdi_shutdown,
};

static struct usb_serial_device_type ftdi_FT232BM_device = {
        .owner =                THIS_MODULE,
        .name =                 "FTDI_FT232BM_Compatible",
        .id_table =             id_table_FT232BM,
        .num_interrupt_in =     0,
        .num_bulk_in =          1,
        .num_bulk_out =         1,
        .num_ports =            1,
        .open =                 ftdi_open,
        .close =                ftdi_close,
        .throttle =             ftdi_throttle,
        .unthrottle =           ftdi_unthrottle,
        .write =                ftdi_write,
        .write_room =           ftdi_write_room,
        .chars_in_buffer =      ftdi_chars_in_buffer,
        .read_bulk_callback =   ftdi_read_bulk_callback,
        .write_bulk_callback =  ftdi_write_bulk_callback,
        .tiocmget =             ftdi_tiocmget,
        .tiocmset =             ftdi_tiocmset,
        .ioctl =                ftdi_ioctl,
```

```
        .set_termios =          ftdi_set_termios,
        .break_ctl =            ftdi_break_ctl,
        .attach =               ftdi_FT232BM_startup,
        .shutdown =             ftdi_shutdown,
};

static struct usb_serial_device_type ftdi_FT2232C_device = {
        .owner =                THIS_MODULE,
        .name =                 "FTDI FT2232C Compatible",
        .id_table =             id_table_FT2232C,
        .num_interrupt_in =     0,
        .num_bulk_in =          1,
        .num_bulk_out =         1,
        .num_ports =            1,
        .open =                 ftdi_open,
        .close =                ftdi_close,
        .throttle =             ftdi_throttle,
        .unthrottle =           ftdi_unthrottle,
        .write =                ftdi_write,
        .write_room =           ftdi_write_room,
        .chars_in_buffer =      ftdi_chars_in_buffer,
        .read_bulk_callback =   ftdi_read_bulk_callback,
        .write_bulk_callback =  ftdi_write_bulk_callback,
        .tiocmget =             ftdi_tiocmget,
        .tiocmset =             ftdi_tiocmset,
        .ioctl =                ftdi_ioctl,
        .set_termios =          ftdi_set_termios,
        .break_ctl =            ftdi_break_ctl,
        .attach =               ftdi_FT2232C_startup,
        .shutdown =             ftdi_shutdown,
};

static struct usb_serial_device_type ftdi_USB_UIRT_device = {
        .owner =                THIS_MODULE,
        .name =                 "USB-UIRT Infrared Tranceiver",
        .id_table =             id_table_USB_UIRT,
        .num_interrupt_in =     0,
        .num_bulk_in =          1,
        .num_bulk_out =         1,
        .num_ports =            1,
        .open =                 ftdi_open,
        .close =                ftdi_close,
        .throttle =             ftdi_throttle,
        .unthrottle =           ftdi_unthrottle,
        .write =                ftdi_write,
        .write_room =           ftdi_write_room,
        .chars_in_buffer =      ftdi_chars_in_buffer,
        .read_bulk_callback =   ftdi_read_bulk_callback,
        .write_bulk_callback =  ftdi_write_bulk_callback,
        .tiocmget =             ftdi_tiocmget,
        .tiocmset =             ftdi_tiocmset,
        .ioctl =                ftdi_ioctl,
        .set_termios =          ftdi_set_termios,
        .break_ctl =            ftdi_break_ctl,
        .attach =               ftdi_USB_UIRT_startup,
        .shutdown =             ftdi_shutdown,
};

/* The TIRA1 is based on a  FT232BM which requires a fixed baud rate of 100000
 * and which requires RTS-CTS to be enabled. */
static struct usb_serial_device_type ftdi_HE_TIRA1_device = {
        .owner =                THIS_MODULE,
        .name =                 "Home-Electronics TIRA-1 IR Transceiver",
        .id_table =             id_table_HE_TIRA1,
        .num_interrupt_in =     0,
        .num_bulk_in =          1,
        .num_bulk_out =         1,
        .num_ports =            1,
        .open =                 ftdi_open,
        .close =                ftdi_close,
        .throttle =             ftdi_throttle,
        .unthrottle =           ftdi_unthrottle,
        .write =                ftdi_write,
        .write_room =           ftdi_write_room,
        .chars_in_buffer =      ftdi_chars_in_buffer,
        .read_bulk_callback =   ftdi_read_bulk_callback,
        .write_bulk_callback =  ftdi_write_bulk_callback,
        .tiocmget =             ftdi_tiocmget,
        .tiocmset =             ftdi_tiocmset,
        .ioctl =                ftdi_ioctl,
        .set_termios =          ftdi_set_termios,
        .break_ctl =            ftdi_break_ctl,
        .attach =               ftdi_HE_TIRA1_startup,
        .shutdown =             ftdi_shutdown,
};


#define WDR_TIMEOUT 5000 /* default urb timeout */

/* High and low are for DTR, RTS etc etc */
#define HIGH 1
```

```
#define LOW 0

/*
 * ***********************************************************************
 * Utlity functions
 * ***********************************************************************
 */

static unsigned short int ftdi_232am_baud_base_to_divisor(int baud, int base)
{
        unsigned short int divisor;
        int divisor3 = base / 2 / baud; // divisor shifted 3 bits to the left
        if ((divisor3 & 0x7) == 7) divisor3 ++; // round x.7/8 up to x+1
        divisor = divisor3 >> 3;
        divisor3 &= 0x7;
        if (divisor3 == 1) divisor |= 0xc000; else // 0.125
        if (divisor3 >= 4) divisor |= 0x4000; else // 0.5
        if (divisor3 != 0) divisor |= 0x8000;      // 0.25
        if (divisor == 1) divisor = 0;  /* special case for maximum baud rate */
        return divisor;
}

static unsigned short int ftdi_232am_baud_to_divisor(int baud)
{
         return(ftdi_232am_baud_base_to_divisor(baud, 48000000));
}

static __u32 ftdi_232bm_baud_base_to_divisor(int baud, int base)
{
        static const unsigned char divfrac[8] = { 0, 3, 2, 4, 1, 5, 6, 7 };
        __u32 divisor;
        int divisor3 = base / 2 / baud; // divisor shifted 3 bits to the left
        divisor = divisor3 >> 3;
        divisor |= (__u32)divfrac[divisor3 & 0x7] << 14;
        /* Deal with special cases for highest baud rates. */
        if (divisor == 1) divisor = 0; else     // 1.0
        if (divisor == 0x4001) divisor = 1;     // 1.5
        return divisor;
}

static __u32 ftdi_232bm_baud_to_divisor(int baud)
{
         return(ftdi_232bm_baud_base_to_divisor(baud, 48000000));
}

static int set_rts(struct usb_serial_port *port, int high_or_low)
{
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        char *buf;
        unsigned ftdi_high_or_low;
        int rv;

        buf = kmalloc(1, GFP_NOIO);
        if (!buf)
                return -ENOMEM;

        if (high_or_low) {
                ftdi_high_or_low = FTDI_SIO_SET_RTS_HIGH;
                priv->last_dtr_rts |= TIOCM_RTS;
        } else {
                ftdi_high_or_low = FTDI_SIO_SET_RTS_LOW;
                priv->last_dtr_rts &= ~TIOCM_RTS;
        }
        rv = usb_control_msg(port->serial->dev,
                             usb_sndctrlpipe(port->serial->dev, 0),
                             FTDI_SIO_SET_MODEM_CTRL_REQUEST,
                             FTDI_SIO_SET_MODEM_CTRL_REQUEST_TYPE,
                             ftdi_high_or_low, priv->interface,
                             buf, 0, WDR_TIMEOUT);

        kfree(buf);
        return rv;
}


static int set_dtr(struct usb_serial_port *port, int high_or_low)
{
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        char *buf;
        unsigned ftdi_high_or_low;
        int rv;

        buf = kmalloc(1, GFP_NOIO);
        if (!buf)
                return -ENOMEM;

        if (high_or_low) {
                ftdi_high_or_low = FTDI_SIO_SET_DTR_HIGH;
                priv->last_dtr_rts |= TIOCM_DTR;
        } else {
                ftdi_high_or_low = FTDI_SIO_SET_DTR_LOW;
                priv->last_dtr_rts &= ~TIOCM_DTR;
```

```
        }
        rv = usb_control_msg(port->serial->dev,
                             usb_sndctrlpipe(port->serial->dev, 0),
                             FTDI_SIO_SET_MODEM_CTRL_REQUEST,
                             FTDI_SIO_SET_MODEM_CTRL_REQUEST_TYPE,
                             ftdi_high_or_low, priv->interface,
                             buf, 0, WDR_TIMEOUT);

        kfree(buf);
        return rv;
}


static __u32 get_ftdi_divisor(struct usb_serial_port * port);


static int change_speed(struct usb_serial_port *port)
{
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        char *buf;
        __u16 urb_value;
        __u16 urb_index;
        __u32 urb_index_value;
        int rv;

        buf = kmalloc(1, GFP_NOIO);
        if (!buf)
                return -ENOMEM;

        urb_index_value = get_ftdi_divisor(port);
        urb_value = (__u16)urb_index_value;
        urb_index = (__u16)(urb_index_value >> 16);
        if (priv->interface) {  /* FT2232C */
                urb_index = (__u16)((urb_index << 8) | priv->interface);
        }

        rv = usb_control_msg(port->serial->dev,
                             usb_sndctrlpipe(port->serial->dev, 0),
                             FTDI_SIO_SET_BAUDRATE_REQUEST,
                             FTDI_SIO_SET_BAUDRATE_REQUEST_TYPE,
                             urb_value, urb_index,
                             buf, 0, 100);

        kfree(buf);
        return rv;
}


static __u32 get_ftdi_divisor(struct usb_serial_port * port)
{ /* get_ftdi_divisor */
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        __u32 div_value = 0;
        int div_okay = 1;
        int baud;

        /*
         * The logic involved in setting the baudrate can be cleanly split in 3 steps.
         * Obtaining the actual baud rate is a little tricky since unix traditionally
         * somehow ignored the possibility to set non-standard baud rates.
         * 1. Standard baud rates are set in tty->termios->c_cflag
         * 2. If these are not enough, you can set any speed using alt_speed as follows:
         *    - set tty->termios->c_cflag speed to B38400
         *    - set your real speed in tty->alt_speed; it gets ignored when
         *      alt_speed==0, (or)
         *    - call TIOCSSERIAL ioctl with (struct serial_struct) set as follows:
         *      flags & ASYNC_SPD_MASK == ASYNC_SPD_[HI, VHI, SHI, WARP], this just
         *      sets alt_speed to (HI: 57600, VHI: 115200, SHI: 230400, WARP: 460800)
         * ** Steps 1, 2 are done courtesy of tty_get_baud_rate
         * 3. You can also set baud rate by setting custom divisor as follows
         *    - set tty->termios->c_cflag speed to B38400
         *    - call TIOCSSERIAL ioctl with (struct serial_struct) set as follows:
         *      o flags & ASYNC_SPD_MASK == ASYNC_SPD_CUST
         *      o custom_divisor set to baud_base / your_new_baudrate
         * ** Step 3 is done courtesy of code borrowed from serial.c - I should really
         *    spend some time and separate+move this common code to serial.c, it is
         *    replicated in nearly every serial driver you see.
         */

        /* 1. Get the baud rate from the tty settings, this observes alt_speed hack */

        baud = tty_get_baud_rate(port->tty);
        dbg("%s - tty_get_baud_rate reports speed %d", __FUNCTION__, baud);

        /* 2. Observe async-compatible custom_divisor hack, update baudrate if needed */

        // MGW: ADDED PER EVOLUTION_RCM MODULE
        if (baud == 230400) /* && port->serial->product == EVO_HYBRID_PID)*/
          {
            baud = 250000;
            dbg("%s: bumped magical 230400 baud to 2.5kb", __FUNCTION__);
          }
        //MGW: End of additions
```

```
        if (baud == 38400 &&
            ((priv->flags & ASYNC_SPD_MASK) == ASYNC_SPD_CUST) &&
             (priv->custom_divisor)) {
                baud = priv->baud_base / priv->custom_divisor;
                dbg("%s - custom_divisor %d sets baud rate to %d", __FUNCTION__, priv->custom_divisor, baud);
        }

        /* 3. Convert baudrate to device-specific divisor */

        if (!baud) baud = 9600;
        switch(priv->chip_type) {
        case SIO: /* SIO chip */
                switch(baud) {
                case 300: div_value = ftdi_sio_b300; break;
                case 600: div_value = ftdi_sio_b600; break;
                case 1200: div_value = ftdi_sio_b1200; break;
                case 2400: div_value = ftdi_sio_b2400; break;
                case 4800: div_value = ftdi_sio_b4800; break;
                case 9600: div_value = ftdi_sio_b9600; break;
                case 19200: div_value = ftdi_sio_b19200; break;
                case 38400: div_value = ftdi_sio_b38400; break;
                case 57600: div_value = ftdi_sio_b57600;  break;
                case 115200: div_value = ftdi_sio_b115200; break;
                } /* baud */
                if (div_value == 0) {
                        dbg("%s - Baudrate (%d) requested is not supported", __FUNCTION__,  baud);
                        div_value = ftdi_sio_b9600;
                        div_okay = 0;
                }
                break;
        case FT8U232AM: /* 8U232AM chip */
                if (baud <= 3000000) {
                        div_value = ftdi_232am_baud_to_divisor(baud);
                } else {
                        dbg("%s - Baud rate too high!", __FUNCTION__);
                        div_value = ftdi_232am_baud_to_divisor(9600);
                        div_okay = 0;
                }
                break;
        case FT232BM: /* FT232BM chip */
        case FT2232C: /* FT2232C chip */
                if (baud <= 3000000) {
                        div_value = ftdi_232bm_baud_to_divisor(baud);
                } else {
                        dbg("%s - Baud rate too high!", __FUNCTION__);
                        div_value = ftdi_232bm_baud_to_divisor(9600);
                        div_okay = 0;
                }
                break;
        } /* priv->chip_type */

        if (div_okay) {
                dbg("%s - Baud rate set to %d (divisor 0x%lX) on chip %s",
                        __FUNCTION__, baud, (unsigned long)div_value,
                        ftdi_chip_name[priv->chip_type]);
        }

        return(div_value);
}


static int get_serial_info(struct usb_serial_port * port, struct serial_struct __user * retinfo)
{
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        struct serial_struct tmp;

        if (!retinfo)
                return -EFAULT;
        memset(&tmp, 0, sizeof(tmp));
        tmp.flags = priv->flags;
        tmp.baud_base = priv->baud_base;
        tmp.custom_divisor = priv->custom_divisor;
        if (copy_to_user(retinfo, &tmp, sizeof(*retinfo)))
                return -EFAULT;
        return 0;
} /* get_serial_info */


static int set_serial_info(struct usb_serial_port * port, struct serial_struct __user * newinfo)
{ /* set_serial_info */
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        struct serial_struct new_serial;
        struct ftdi_private old_priv;

        if (copy_from_user(&new_serial, newinfo, sizeof(new_serial)))
                return -EFAULT;
        old_priv = * priv;

        /* Do error checking and permission checking */

        if (!capable(CAP_SYS_ADMIN)) {
```

```
                if (((new_serial.flags & ~ASYNC_USR_MASK) !=
                    (priv->flags & ~ASYNC_USR_MASK)))
                        return -EPERM;
                priv->flags = ((priv->flags & ~ASYNC_USR_MASK) |
                               (new_serial.flags & ASYNC_USR_MASK));
                priv->custom_divisor = new_serial.custom_divisor;
                goto check_and_exit;
        }

        if ((new_serial.baud_base != priv->baud_base) &&
            (new_serial.baud_base < 9600))
                return -EINVAL;

        /* Make the changes - these are privileged changes! */

        priv->flags = ((priv->flags & ~ASYNC_FLAGS) |
                       (new_serial.flags & ASYNC_FLAGS));
        priv->custom_divisor = new_serial.custom_divisor;

        port->tty->low_latency = (priv->flags & ASYNC_LOW_LATENCY) ? 1 : 0;

check_and_exit:
        if ((old_priv.flags & ASYNC_SPD_MASK) !=
            (priv->flags & ASYNC_SPD_MASK)) {
                if ((priv->flags & ASYNC_SPD_MASK) == ASYNC_SPD_HI)
                        port->tty->alt_speed = 57600;
                else if ((priv->flags & ASYNC_SPD_MASK) == ASYNC_SPD_VHI)
                        port->tty->alt_speed = 115200;
                else if ((priv->flags & ASYNC_SPD_MASK) == ASYNC_SPD_SHI)
                        port->tty->alt_speed = 230400;
                else if ((priv->flags & ASYNC_SPD_MASK) == ASYNC_SPD_WARP)
                        port->tty->alt_speed = 460800;
                else
                        port->tty->alt_speed = 0;
        }
        if (((old_priv.flags & ASYNC_SPD_MASK) !=
             (priv->flags & ASYNC_SPD_MASK)) ||
            (((priv->flags & ASYNC_SPD_MASK) == ASYNC_SPD_CUST) &&
             (old_priv.custom_divisor != priv->custom_divisor))) {
                change_speed(port);
        }

        return (0);

} /* set_serial_info */


/*
 * ***********************************************************************
 * Sysfs Attribute
 * ***********************************************************************
 */

static ssize_t show_latency_timer(struct device *dev, char *buf)
{
        struct usb_serial_port *port = to_usb_serial_port(dev);
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        struct usb_device *udev;
        unsigned short latency = 0;
        int rv = 0;

        udev = to_usb_device(dev);

        dbg("%s",__FUNCTION__);

        rv = usb_control_msg(udev,
                             usb_rcvctrlpipe(udev, 0),
                             FTDI_SIO_GET_LATENCY_TIMER_REQUEST,
                             FTDI_SIO_GET_LATENCY_TIMER_REQUEST_TYPE,
                             0, priv->interface,
                             (char*) &latency, 1, WDR_TIMEOUT);

        if (rv < 0) {
                dev_err(dev, "Unable to read latency timer: %i", rv);
                return -EIO;
        }
        return sprintf(buf, "%i\n", latency);
}

/* Write a new value of the latency timer, in units of milliseconds. */
static ssize_t store_latency_timer(struct device *dev, const char *valbuf,
                                   size_t count)
{
        struct usb_serial_port *port = to_usb_serial_port(dev);
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        struct usb_device *udev;
        char buf[1];
        int v = simple_strtoul(valbuf, NULL, 10);
        int rv = 0;

        udev = to_usb_device(dev);
```

```
        dbg("%s: setting latency timer = %i", __FUNCTION__, v);

        rv = usb_control_msg(udev,
                             usb_sndctrlpipe(udev, 0),
                             FTDI_SIO_SET_LATENCY_TIMER_REQUEST,
                             FTDI_SIO_SET_LATENCY_TIMER_REQUEST_TYPE,
                             v, priv->interface,
                             buf, 0, WDR_TIMEOUT);

        if (rv < 0) {
                dev_err(dev, "Unable to write latency timer: %i", rv);
                return -EIO;
        }

        return count;
}

/* Write an event character directly to the FTDI register.  The ASCII
   value is in the low 8 bits, with the enable bit in the 9th bit. */
static ssize_t store_event_char(struct device *dev, const char *valbuf,
                                size_t count)
{
        struct usb_serial_port *port = to_usb_serial_port(dev);
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        struct usb_device *udev;
        char buf[1];
        int v = simple_strtoul(valbuf, NULL, 10);
        int rv = 0;

        udev = to_usb_device(dev);

        dbg("%s: setting event_char = %i", __FUNCTION__, v);

        rv = usb_control_msg(udev,
                             usb_sndctrlpipe(udev, 0),
                             FTDI_SIO_SET_EVENT_CHAR_REQUEST,
                             FTDI_SIO_SET_EVENT_CHAR_REQUEST_TYPE,
                             v, priv->interface,
                             buf, 0, WDR_TIMEOUT);

        if (rv < 0) {
                dbg("Unable to write event character: %i", rv);
                return -EIO;
        }

        return count;
}

static DEVICE_ATTR(latency_timer, S_IWUSR | S_IRUGO, show_latency_timer, store_latency_timer);
static DEVICE_ATTR(event_char, S_IWUSR, NULL, store_event_char);

static void create_sysfs_attrs(struct usb_serial *serial)
{
        struct ftdi_private *priv;
        struct usb_device *udev;

        dbg("%s",__FUNCTION__);

        priv = usb_get_serial_port_data(serial->port[0]);
        udev = serial->dev;

        /* XXX I've no idea if the original SIO supports the event_char
         * sysfs parameter, so I'm playing it safe.  */
        if (priv->chip_type != SIO) {
                dbg("sysfs attributes for %s", ftdi_chip_name[priv->chip_type]);
                device_create_file(&udev->dev, &dev_attr_event_char);
                if (priv->chip_type == FT232BM || priv->chip_type == FT2232C) {
                        device_create_file(&udev->dev, &dev_attr_latency_timer);
                }
        }
}

static void remove_sysfs_attrs(struct usb_serial *serial)
{
        struct ftdi_private *priv;
        struct usb_device *udev;

        dbg("%s",__FUNCTION__);

        priv = usb_get_serial_port_data(serial->port[0]);
        udev = serial->dev;

        /* XXX see create_sysfs_attrs */
        if (priv->chip_type != SIO) {
                device_remove_file(&udev->dev, &dev_attr_event_char);
                if (priv->chip_type == FT232BM || priv->chip_type == FT2232C) {
                        device_remove_file(&udev->dev, &dev_attr_latency_timer);
                }
        }

}
```

```
/*
 * ************************************************************************
 * FTDI driver specific functions
 * ************************************************************************
 */

/* Common startup subroutine */
/* Called from ftdi_SIO_startup, etc. */
static int ftdi_common_startup (struct usb_serial *serial)
{
        struct usb_serial_port *port = serial->port[0];
        struct ftdi_private *priv;

        dbg("%s",__FUNCTION__);

        priv = kmalloc(sizeof(struct ftdi_private), GFP_KERNEL);
        if (!priv){
                err("%s-kmalloc(%Zd)failed.", __FUNCTION__, sizeof(struct ftdi_private));
                return -ENOMEM;
        }
        memset(priv, 0, sizeof(*priv));

        spin_lock_init(&priv->rx_lock);
        init_waitqueue_head(&priv->delta_msr_wait);
        /* This will push the characters through immediately rather
           than queue a task to deliver them */
        priv->flags = ASYNC_LOW_LATENCY;

        /* Increase the size of read buffers */
        kfree(port->bulk_in_buffer);
        port->bulk_in_buffer = kmalloc (BUFSZ, GFP_KERNEL);
        if (!port->bulk_in_buffer) {
                kfree (priv);
                return -ENOMEM;
        }
        if (port->read_urb) {
                port->read_urb->transfer_buffer = port->bulk_in_buffer;
                port->read_urb->transfer_buffer_length = BUFSZ;
        }

        INIT_WORK(&priv->rx_work, ftdi_process_read, port);

        /* Free port's existing write urb and transfer buffer. */
        if (port->write_urb) {
                usb_free_urb (port->write_urb);
                port->write_urb = NULL;
        }
        kfree(port->bulk_out_buffer);
        port->bulk_out_buffer = NULL;

        usb_set_serial_port_data(serial->port[0], priv);

        return (0);
}


/* Startup for the SIO chip */
/* Called from usbserial:serial_probe */
static int ftdi_SIO_startup (struct usb_serial *serial)
{
        struct ftdi_private *priv;
        int err;

        dbg("%s",__FUNCTION__);

        err = ftdi_common_startup(serial);
        if (err){
                return (err);
        }

        priv = usb_get_serial_port_data(serial->port[0]);
        priv->chip_type = SIO;
        priv->baud_base = 12000000 / 16;
        priv->write_offset = 1;

        return (0);
}

/* Startup for the 8U232AM chip */
/* Called from usbserial:serial_probe */
static int ftdi_8U232AM_startup (struct usb_serial *serial)
{ /* ftdi_8U232AM_startup */
        struct ftdi_private *priv;
        int err;

        dbg("%s",__FUNCTION__);
        err = ftdi_common_startup(serial);
        if (err){
                return (err);
        }

        priv = usb_get_serial_port_data(serial->port[0]);
```

```
        priv->chip_type = FT8U232AM;
        priv->baud_base = 48000000 / 2; /* Would be / 16, but FTDI supports 0.125, 0.25 and 0.5 divisor fractions! */

        create_sysfs_attrs(serial);

        return (0);
} /* ftdi_8U232AM_startup */

/* Startup for the FT232BM chip */
/* Called from usbserial:serial_probe */
static int ftdi_FT232BM_startup (struct usb_serial *serial)
{ /* ftdi_FT232BM_startup */
        struct ftdi_private *priv;
        int err;

        dbg("%s",__FUNCTION__);
        err = ftdi_common_startup(serial);
        if (err){
                return (err);
        }

        priv = usb_get_serial_port_data(serial->port[0]);
        priv->chip_type = FT232BM;
        priv->baud_base = 48000000 / 2; /* Would be / 16, but FT232BM supports multiple of 0.125 divisor fractions! */

        create_sysfs_attrs(serial);

        return (0);
} /* ftdi_FT232BM_startup */

/* Startup for the FT2232C chip */
/* Called from usbserial:serial_probe */
static int ftdi_FT2232C_startup (struct usb_serial *serial)
{ /* ftdi_FT2232C_startup */
        struct ftdi_private *priv;
        int err;
        int inter;

        dbg("%s",__FUNCTION__);
        err = ftdi_common_startup(serial);
        if (err){
                return (err);
        }

        priv = usb_get_serial_port_data(serial->port[0]);
        priv->chip_type = FT2232C;
        inter = serial->interface->altsetting->desc.bInterfaceNumber;

        if (inter) {
                priv->interface = PIT_SIOB;
        }
        else  {
                priv->interface = PIT_SIOA;
        }
        priv->baud_base = 48000000 / 2; /* Would be / 16, but FT2232C supports multiple of 0.125 divisor fractions! */

        create_sysfs_attrs(serial);

        return (0);
} /* ftdi_FT2232C_startup */

/* Startup for the USB-UIRT device, which requires hardwired baudrate (38400 gets mapped to 312500) */
/* Called from usbserial:serial_probe */
static int ftdi_USB_UIRT_startup (struct usb_serial *serial)
{ /* ftdi_USB_UIRT_startup */
        struct ftdi_private *priv;
        int err;

        dbg("%s",__FUNCTION__);
        err = ftdi_8U232AM_startup(serial);
        if (err){
                return (err);
        }

        priv = usb_get_serial_port_data(serial->port[0]);
        priv->flags |= ASYNC_SPD_CUST;
        priv->custom_divisor = 77;
        priv->force_baud = B38400;

        return (0);
} /* ftdi_USB_UIRT_startup */

/* Startup for the HE-TIRA1 device, which requires hardwired
 * baudrate (38400 gets mapped to 100000) */
static int ftdi_HE_TIRA1_startup (struct usb_serial *serial)
{ /* ftdi_HE_TIRA1_startup */
        struct ftdi_private *priv;
        int err;

        dbg("%s",__FUNCTION__);
        err = ftdi_FT232BM_startup(serial);
        if (err){
```

```
                return (err);
        }

        priv = usb_get_serial_port_data(serial->port[0]);
        priv->flags |= ASYNC_SPD_CUST;
        priv->custom_divisor = 240;
        priv->force_baud = B38400;
        priv->force_rtscts = 1;

        return (0);
} /* ftdi_HE_TIRA1_startup */


/* ftdi_shutdown is called from usbserial:usb_serial_disconnect
 *   it is called when the usb device is disconnected
 *
 *   usbserial:usb_serial_disconnect
 *      calls __serial_close for each open of the port
 *      shutdown is called then (ie ftdi_shutdown)
 */


static void ftdi_shutdown (struct usb_serial *serial)
{ /* ftdi_shutdown */

        struct usb_serial_port *port = serial->port[0];
        struct ftdi_private *priv = usb_get_serial_port_data(port);

        dbg("%s", __FUNCTION__);

        remove_sysfs_attrs(serial);

        /* all open ports are closed at this point
         *    (by usbserial.c:__serial_close, which calls ftdi_close)
         */

        if (priv) {
                usb_set_serial_port_data(port, NULL);
                kfree(priv);
        }
} /* ftdi_shutdown */


static int  ftdi_open (struct usb_serial_port *port, struct file *filp)
{ /* ftdi_open */
        struct termios tmp_termios;
        struct usb_device *dev = port->serial->dev;
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        unsigned long flags;

        int result = 0;
        char buf[1]; /* Needed for the usb_control_msg I think */

        dbg("%s", __FUNCTION__);


        port->tty->low_latency = (priv->flags & ASYNC_LOW_LATENCY) ? 1 : 0;

        /* No error checking for this (will get errors later anyway) */
        /* See ftdi_sio.h for description of what is reset */
        usb_control_msg(dev, usb_sndctrlpipe(dev, 0),
                        FTDI_SIO_RESET_REQUEST, FTDI_SIO_RESET_REQUEST_TYPE,
                        FTDI_SIO_RESET_SIO,
                        priv->interface, buf, 0, WDR_TIMEOUT);

        /* Termios defaults are set by usb_serial_init. We don't change
           port->tty->termios - this would loose speed settings, etc.
           This is same behaviour as serial.c/rs_open() - Kuba */

        /* ftdi_set_termios  will send usb control messages */
        ftdi_set_termios(port, &tmp_termios);

        /* FIXME: Flow control might be enabled, so it should be checked -
           we have no control of defaults! */
        /* Turn on RTS and DTR since we are not flow controlling by default */
        if (set_dtr(port, HIGH) < 0) {
                err("%s Error from DTR HIGH urb", __FUNCTION__);
        }
        if (set_rts(port, HIGH) < 0){
                err("%s Error from RTS HIGH urb", __FUNCTION__);
        }

        /* Not throttled */
        spin_lock_irqsave(&priv->rx_lock, flags);
        priv->rx_flags &= ~(THROTTLED | ACTUALLY_THROTTLED);
        spin_unlock_irqrestore(&priv->rx_lock, flags);

        /* Start reading from the device */
        priv->rx_processed = 0;
        usb_fill_bulk_urb(port->read_urb, dev,
                      usb_rcvbulkpipe(dev, port->bulk_in_endpointAddress),
                      port->read_urb->transfer_buffer, port->read_urb->transfer_buffer_length,
```

```
                        ftdi_read_bulk_callback, port);
        result = usb_submit_urb(port->read_urb, GFP_KERNEL);
        if (result)
                err("%s - failed submitting read urb, error %d", __FUNCTION__, result);


        return result;
} /* ftdi_open */



/*
 * usbserial: __serial_close  only calls ftdi_close if the point is open
 *
 *   This only gets called when it is the last close
 *
 *
 */

static void ftdi_close (struct usb_serial_port *port, struct file *filp)
{ /* ftdi_close */
        unsigned int c_cflag = port->tty->termios->c_cflag;
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        char buf[1];

        dbg("%s", __FUNCTION__);

        if (c_cflag & HUPCL){
                /* Disable flow control */
                if (usb_control_msg(port->serial->dev,
                                    usb_sndctrlpipe(port->serial->dev, 0),
                                    FTDI_SIO_SET_FLOW_CTRL_REQUEST,
                                    FTDI_SIO_SET_FLOW_CTRL_REQUEST_TYPE,
                                    0, priv->interface, buf, 0,
                                    WDR_TIMEOUT) < 0) {
                        err("error from flowcontrol urb");
                }

                /* drop DTR */
                if (set_dtr(port, LOW) < 0){
                        err("Error from DTR LOW urb");
                }
                /* drop RTS */
                if (set_rts(port, LOW) < 0) {
                        err("Error from RTS LOW urb");
                }
        } /* Note change no line if hupcl is off */

        /* cancel any scheduled reading */
        cancel_delayed_work(&priv->rx_work);
        flush_scheduled_work();

        /* shutdown our bulk read */
        if (port->read_urb)
                usb_kill_urb(port->read_urb);
} /* ftdi_close */



/* The SIO requires the first byte to have:
 *  B0 1
 *  B1 0
 *  B2..7 length of message excluding byte 0
 *
 * The new devices do not require this byte
 */
static int ftdi_write (struct usb_serial_port *port,
                       const unsigned char *buf, int count)
{ /* ftdi_write */
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        struct urb *urb;
        unsigned char *buffer;
        int data_offset ;       /* will be 1 for the SIO and 0 otherwise */
        int status;
        int transfer_size;

        dbg("%s port %d, %d bytes", __FUNCTION__, port->number, count);

        if (count == 0) {
                dbg("write request of 0 bytes");
                return 0;
        }

        data_offset = priv->write_offset;
        dbg("data_offset set to %d",data_offset);

        /* Determine total transfer size */
        transfer_size = count;
        if (data_offset > 0) {
                /* Original sio needs control bytes too... */
                transfer_size += (data_offset *
                                  ((count + (PKTSZ - 1 - data_offset)) /
```

```
                                (PKTSZ - data_offset)));
        }

        buffer = kmalloc (transfer_size, GFP_ATOMIC);
        if (!buffer) {
                err("%s ran out of kernel memory for urb ...", __FUNCTION__);
                return -ENOMEM;
        }

        urb = usb_alloc_urb(0, GFP_ATOMIC);
        if (!urb) {
                err("%s - no more free urbs", __FUNCTION__);
                kfree (buffer);
                return -ENOMEM;
        }

        /* Copy data */
        if (data_offset > 0) {
                /* Original sio requires control byte at start of each packet. */
                int user_pktsz = PKTSZ - data_offset;
                int todo = count;
                unsigned char *first_byte = buffer;
                const unsigned char *current_position = buf;

                while (todo > 0) {
                        if (user_pktsz > todo) {
                                user_pktsz = todo;
                        }
                        /* Write the control byte at the front of the packet*/
                        *first_byte = 1 | ((user_pktsz) << 2);
                        /* Copy data for packet */
                        memcpy (first_byte + data_offset,
                                current_position, user_pktsz);
                        first_byte += user_pktsz + data_offset;
                        current_position += user_pktsz;
                        todo -= user_pktsz;
                }
        } else {
                /* No control byte required. */
                /* Copy in the data to send */
                memcpy (buffer, buf, count);
        }

        usb_serial_debug_data(debug, &port->dev, __FUNCTION__, transfer_size, buffer);

        /* fill the buffer and send it */
        usb_fill_bulk_urb(urb, port->serial->dev,
                        usb_sndbulkpipe(port->serial->dev, port->bulk_out_endpointAddress),
                        buffer, transfer_size,
                        ftdi_write_bulk_callback, port);

        status = usb_submit_urb(urb, GFP_ATOMIC);
        if (status) {
                err("%s - failed submitting write urb, error %d", __FUNCTION__, status);
                count = status;
                kfree (buffer);
        }

        /* we are done with this urb, so let the host driver
         * really free it when it is finished with it */
        usb_free_urb (urb);

        dbg("%s write returning: %d", __FUNCTION__, count);
        return count;
} /* ftdi_write */


/* This function may get called when the device is closed */

static void ftdi_write_bulk_callback (struct urb *urb, struct pt_regs *regs)
{
        struct usb_serial_port *port = (struct usb_serial_port *)urb->context;

        /* free up the transfer buffer, as usb_free_urb() does not do this */
        kfree (urb->transfer_buffer);

        dbg("%s - port %d", __FUNCTION__, port->number);

        if (urb->status) {
                dbg("nonzero write bulk status received: %d", urb->status);
                return;
        }

        schedule_work(&port->work);
} /* ftdi_write_bulk_callback */


static int ftdi_write_room( struct usb_serial_port *port )
{
        dbg("%s - port %d", __FUNCTION__, port->number);

        /*
```

```
         * We really can take anything the user throws at us
         * but let's pick a nice big number to tell the tty
         * layer that we have lots of free space
         */
        return 2048;
} /* ftdi_write_room */


static int ftdi_chars_in_buffer (struct usb_serial_port *port)
{ /* ftdi_chars_in_buffer */
        dbg("%s - port %d", __FUNCTION__, port->number);

        /*
         * We can't really account for how much data we
         * have sent out, but hasn't made it through to the
         * device, so just tell the tty layer that everything
         * is flushed.
         */
        return 0;
} /* ftdi_chars_in_buffer */



static void ftdi_read_bulk_callback (struct urb *urb, struct pt_regs *regs)
{ /* ftdi_read_bulk_callback */
        struct usb_serial_port *port = (struct usb_serial_port *)urb->context;
        struct tty_struct *tty;
        struct ftdi_private *priv;

        if (urb->number_of_packets > 0) {
                err("%s transfer_buffer_length %d actual_length %d number_of_packets %d",__FUNCTION__,
                    urb->transfer_buffer_length, urb->actual_length, urb->number_of_packets );
                err("%s transfer_flags %x ", __FUNCTION__,urb->transfer_flags );
        }

        dbg("%s - port %d", __FUNCTION__, port->number);

        if (port->open_count <= 0)
                return;

        tty = port->tty;
        if (!tty) {
                dbg("%s - bad tty pointer - exiting",__FUNCTION__);
                return;
        }

        priv = usb_get_serial_port_data(port);
        if (!priv) {
                dbg("%s - bad port private data pointer - exiting", __FUNCTION__);
                return;
        }

        if (urb != port->read_urb) {
                err("%s - Not my urb!", __FUNCTION__);
        }

        if (urb->status) {
                /* This will happen at close every time so it is a dbg not an err */
                dbg("(this is ok on close) nonzero read bulk status received: %d", urb->status);
                return;
        }

        ftdi_process_read(port);

} /* ftdi_read_bulk_callback */


static void ftdi_process_read (void *param)
{ /* ftdi_process_read */
        struct usb_serial_port *port = (struct usb_serial_port*)param;
        struct urb *urb;
        struct tty_struct *tty;
        struct ftdi_private *priv;
        char error_flag;
        unsigned char *data;

        int i;
        int result;
        int need_flip;
        int packet_offset;
        unsigned long flags;

        dbg("%s - port %d", __FUNCTION__, port->number);

        if (port->open_count <= 0)
                return;

        tty = port->tty;
        if (!tty) {
                dbg("%s - bad tty pointer - exiting",__FUNCTION__);
                return;
        }
```

```
priv = usb_get_serial_port_data(port);
if (!priv) {
        dbg("%s - bad port private data pointer - exiting", __FUNCTION__);
        return;
}

urb = port->read_urb;
if (!urb) {
        dbg("%s - bad read urb pointer - exiting", __FUNCTION__);
        return;
}

data = urb->transfer_buffer;

if (priv->rx_processed) {
        dbg("%s - already processed: %d bytes, %d remain", __FUNCTION__,
                        priv->rx_processed,
                        urb->actual_length - priv->rx_processed);
} else {
        /* The first two bytes of every read packet are status */
        if (urb->actual_length > 2) {
                usb_serial_debug_data(debug, &port->dev, __FUNCTION__, urb->actual_length, data);
        } else {
                dbg("Status only: %03oo %03oo",data[0],data[1]);
        }
}


/* TO DO -- check for hung up line and handle appropriately: */
/*   send hangup  */
/* See acm.c - you do a tty_hangup  - eg tty_hangup(tty) */
/* if CD is dropped and the line is not CLOCAL then we should hangup */

need_flip = 0;
for (packet_offset = priv->rx_processed; packet_offset < urb->actual_length; packet_offset += PKTSZ) {
        int length;

        /* Compare new line status to the old one, signal if different */
        /* N.B. packet may be processed more than once, but differences
         * are only processed once.  */
        if (priv != NULL) {
                char new_status = data[packet_offset+0] & FTDI_STATUS_B0_MASK;
                if (new_status != priv->prev_status) {
                        priv->diff_status |= new_status ^ priv->prev_status;
                        wake_up_interruptible(&priv->delta_msr_wait);
                        priv->prev_status = new_status;
                }
        }

        length = min(PKTSZ, urb->actual_length-packet_offset)-2;
        if (length < 0) {
                err("%s - bad packet length: %d", __FUNCTION__, length+2);
                length = 0;
        }

        /* have to make sure we don't overflow the buffer
           with tty_insert_flip_char's */
        if (tty->flip.count+length > TTY_FLIPBUF_SIZE) {
                tty_flip_buffer_push(tty);
                need_flip = 0;

                if (tty->flip.count != 0) {
                        /* flip didn't work, this happens when ftdi_process_read() is
                         * called from ftdi_unthrottle, because TTY_DONT_FLIP is set */
                        dbg("%s - flip buffer push failed", __FUNCTION__);
                        break;
                }
        }
        if (priv->rx_flags & THROTTLED) {
                dbg("%s - throttled", __FUNCTION__);
                break;
        }
        if (tty->ldisc.receive_room(tty)-tty->flip.count < length) {
                /* break out & wait for throttling/unthrottling to happen */
                dbg("%s - receive room low", __FUNCTION__);
                break;
        }

        /* Handle errors and break */
        error_flag = TTY_NORMAL;
        /* Although the device uses a bitmask and hence can have multiple */
        /* errors on a packet - the order here sets the priority the */
        /* error is returned to the tty layer  */

        if ( data[packet_offset+1] & FTDI_RS_OE ) {
                error_flag = TTY_OVERRUN;
                dbg("OVERRRUN error");
        }
        if ( data[packet_offset+1] & FTDI_RS_BI ) {
                error_flag = TTY_BREAK;
                dbg("BREAK received");
```

```
                }
                if ( data[packet_offset+1] & FTDI_RS_PE ) {
                        error_flag = TTY_PARITY;
                        dbg("PARITY error");
                }
                if ( data[packet_offset+1] & FTDI_RS_FE ) {
                        error_flag = TTY_FRAME;
                        dbg("FRAMING error");
                }
                if (length > 0) {
                        for (i = 2; i < length+2; i++) {
                                /* Note that the error flag is duplicated for
                                   every character received since we don't know
                                   which character it applied to */
                                tty_insert_flip_char(tty, data[packet_offset+i], error_flag);
                        }
                        need_flip = 1;
                }

#ifdef NOT_CORRECT_BUT_KEEPING_IT_FOR_NOW
                /* if a parity error is detected you get status packets forever
                   until a character is sent without a parity error.
                   This doesn't work well since the application receives a never
                   ending stream of bad data - even though new data hasn't been sent.
                   Therefore I (bill) have taken this out.
                   However - this might make sense for framing errors and so on
                   so I am leaving the code in for now.
                */
                else {
                        if (error_flag != TTY_NORMAL){
                                dbg("error_flag is not normal");
                                /* In this case it is just status - if that is an error send a bad character */
                                if(tty->flip.count >= TTY_FLIPBUF_SIZE) {
                                        tty_flip_buffer_push(tty);
                                }
                                tty_insert_flip_char(tty, 0xff, error_flag);
                                need_flip = 1;
                        }
                }
#endif
        } /* "for(packet_offset=0..." */

        /* Low latency */
        if (need_flip) {
                tty_flip_buffer_push(tty);
        }

        if (packet_offset < urb->actual_length) {
                /* not completely processed - record progress */
                priv->rx_processed = packet_offset;
                dbg("%s - incomplete, %d bytes processed, %d remain",
                                __FUNCTION__, packet_offset,
                                urb->actual_length - packet_offset);
                /* check if we were throttled while processing */
                spin_lock_irqsave(&priv->rx_lock, flags);
                if (priv->rx_flags & THROTTLED) {
                        priv->rx_flags |= ACTUALLY_THROTTLED;
                        spin_unlock_irqrestore(&priv->rx_lock, flags);
                        dbg("%s - deferring remainder until unthrottled",
                                        __FUNCTION__);
                        return;
                }
                spin_unlock_irqrestore(&priv->rx_lock, flags);
                /* if the port is closed stop trying to read */
                if (port->open_count > 0){
                        /* delay processing of remainder */
                        schedule_delayed_work(&priv->rx_work, 1);
                } else {
                        dbg("%s - port is closed", __FUNCTION__);
                }
                return;
        }

        /* urb is completely processed */
        priv->rx_processed = 0;

        /* if the port is closed stop trying to read */
        if (port->open_count > 0){
                /* Continue trying to always read  */
                usb_fill_bulk_urb(port->read_urb, port->serial->dev,
                                usb_rcvbulkpipe(port->serial->dev, port->bulk_in_endpointAddress),
                                port->read_urb->transfer_buffer, port->read_urb->transfer_buffer_length,
                                ftdi_read_bulk_callback, port);

                result = usb_submit_urb(port->read_urb, GFP_ATOMIC);
                if (result)
                        err("%s - failed resubmitting read urb, error %d", __FUNCTION__, result);
        }

        return;
} /* ftdi_process_read */
```

```
static void ftdi_break_ctl( struct usb_serial_port *port, int break_state )
{
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        __u16 urb_value = 0;
        char buf[1];

        /* break_state = -1 to turn on break, and 0 to turn off break */
        /* see drivers/char/tty_io.c to see it used */
        /* last_set_data_urb_value NEVER has the break bit set in it */

        if (break_state) {
                urb_value = priv->last_set_data_urb_value | FTDI_SIO_SET_BREAK;
        } else {
                urb_value = priv->last_set_data_urb_value;
        }


        if (usb_control_msg(port->serial->dev, usb_sndctrlpipe(port->serial->dev, 0),
                            FTDI_SIO_SET_DATA_REQUEST,
                            FTDI_SIO_SET_DATA_REQUEST_TYPE,
                            urb_value , priv->interface,
                            buf, 0, WDR_TIMEOUT) < 0) {
                err("%s FAILED to enable/disable break state (state was %d)", __FUNCTION__, break_state);
        }

        dbg("%s break state is %d - urb is %d", __FUNCTION__, break_state, urb_value);

}


/* old_termios contains the original termios settings and tty->termios contains
 * the new setting to be used
 * WARNING: set_termios calls this with old_termios in kernel space
 */

static void ftdi_set_termios (struct usb_serial_port *port, struct termios *old_termios)
{ /* ftdi_termios */
        struct usb_device *dev = port->serial->dev;
        unsigned int cflag = port->tty->termios->c_cflag;
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        __u16 urb_value; /* will hold the new flags */
        char buf[1]; /* Perhaps I should dynamically alloc this? */

        // Added for xon/xoff support
        unsigned int iflag = port->tty->termios->c_iflag;
        unsigned char vstop;
        unsigned char vstart;

        dbg("%s", __FUNCTION__);

        /* Force baud rate if this device requires it, unless it is set to B0. */
        if (priv->force_baud && ((port->tty->termios->c_cflag & CBAUD) != B0)) {
                dbg("%s: forcing baud rate for this device", __FUNCTION__);
                port->tty->termios->c_cflag &= ~CBAUD;
                port->tty->termios->c_cflag |= priv->force_baud;
        }

        /* Force RTS-CTS if this device requires it. */
        if (priv->force_rtscts) {
                dbg("%s: forcing rtscts for this device", __FUNCTION__);
                port->tty->termios->c_cflag |= CRTSCTS;
        }

        cflag = port->tty->termios->c_cflag;

        /* FIXME -For this cut I don't care if the line is really changing or
           not - so just do the change regardless - should be able to
           compare old_termios and tty->termios */
        /* NOTE These routines can get interrupted by
           ftdi_sio_read_bulk_callback - need to examine what this
           means - don't see any problems yet */

        /* Set number of data bits, parity, stop bits */

        urb_value = 0;
        urb_value |= (cflag & CSTOPB ? FTDI_SIO_SET_DATA_STOP_BITS_2 :
                      FTDI_SIO_SET_DATA_STOP_BITS_1);
        urb_value |= (cflag & PARENB ?
                      (cflag & PARODD ? FTDI_SIO_SET_DATA_PARITY_ODD :
                       FTDI_SIO_SET_DATA_PARITY_EVEN) :
                      FTDI_SIO_SET_DATA_PARITY_NONE);
        if (cflag & CSIZE) {
                switch (cflag & CSIZE) {
                case CS5: urb_value |= 5; dbg("Setting CS5"); break;
                case CS6: urb_value |= 6; dbg("Setting CS6"); break;
                case CS7: urb_value |= 7; dbg("Setting CS7"); break;
                case CS8: urb_value |= 8; dbg("Setting CS8"); break;
                default:
                        err("CSIZE was set but not CS5-CS8");
                }
        }
```

```
/* This is needed by the break command since it uses the same command - but is
 *  or'ed with this value  */
priv->last_set_data_urb_value = urb_value;

if (usb_control_msg(dev, usb_sndctrlpipe(dev, 0),
                    FTDI_SIO_SET_DATA_REQUEST,
                    FTDI_SIO_SET_DATA_REQUEST_TYPE,
                    urb_value , priv->interface,
                    buf, 0, 100) < 0) {
        err("%s FAILED to set databits/stopbits/parity", __FUNCTION__);
}

/* Now do the baudrate */
if ((cflag & CBAUD) == B0 ) {
        /* Disable flow control */
        if (usb_control_msg(dev, usb_sndctrlpipe(dev, 0),
                            FTDI_SIO_SET_FLOW_CTRL_REQUEST,
                            FTDI_SIO_SET_FLOW_CTRL_REQUEST_TYPE,
                            0, priv->interface,
                            buf, 0, WDR_TIMEOUT) < 0) {
                err("%s error from disable flowcontrol urb", __FUNCTION__);
        }
        /* Drop RTS and DTR */
        if (set_dtr(port, LOW) < 0){
                err("%s Error from DTR LOW urb", __FUNCTION__);
        }
        if (set_rts(port, LOW) < 0){
                err("%s Error from RTS LOW urb", __FUNCTION__);
        }

} else {
        /* set the baudrate determined before */
        if (change_speed(port)) {
                err("%s urb failed to set baurdrate", __FUNCTION__);
        }
        /* Ensure  RTS and DTR are raised */
        else if (set_dtr(port, HIGH) < 0){
                err("%s Error from DTR HIGH urb", __FUNCTION__);
        }
        else if (set_rts(port, HIGH) < 0){
                err("%s Error from RTS HIGH urb", __FUNCTION__);
        }
}

/* Set flow control */
/* Note device also supports DTR/CD (ugh) and Xon/Xoff in hardware */
if (cflag & CRTSCTS) {
        dbg("%s Setting to CRTSCTS flow control", __FUNCTION__);
        if (usb_control_msg(dev,
                            usb_sndctrlpipe(dev, 0),
                            FTDI_SIO_SET_FLOW_CTRL_REQUEST,
                            FTDI_SIO_SET_FLOW_CTRL_REQUEST_TYPE,
                            0 , (FTDI_SIO_RTS_CTS_HS | priv->interface),
                            buf, 0, WDR_TIMEOUT) < 0) {
                err("urb failed to set to rts/cts flow control");
        }

} else {
        /*
         * Xon/Xoff code
         *
         * Check the IXOFF status in the iflag component of the termios structure
         * if IXOFF is not set, the pre-xon/xoff code is executed.
        */
        if (iflag & IXOFF) {
                dbg("%s  request to enable xonxoff iflag=%04x",__FUNCTION__,iflag);
                // Try to enable the XON/XOFF on the ftdi_sio
                // Set the vstart and vstop -- could have been done up above where
                // a lot of other dereferencing is done but that would be very
                // inefficient as vstart and vstop are not always needed
                vstart=port->tty->termios->c_cc[VSTART];
                vstop=port->tty->termios->c_cc[VSTOP];
                urb_value=(vstop << 8) | (vstart);

                if (usb_control_msg(dev,
                                    usb_sndctrlpipe(dev, 0),
                                    FTDI_SIO_SET_FLOW_CTRL_REQUEST,
                                    FTDI_SIO_SET_FLOW_CTRL_REQUEST_TYPE,
                                    urb_value , (FTDI_SIO_XON_XOFF_HS
                                                 | priv->interface),
                                    buf, 0, WDR_TIMEOUT) < 0) {
                        err("urb failed to set to xon/xoff flow control");
                }
        } else {
                /* else clause to only run if cfag ! CRTSCTS and iflag ! XOFF */
                /* CHECKME Assuming XON/XOFF handled by tty stack - not by device */
                dbg("%s Turning off hardware flow control", __FUNCTION__);
                if (usb_control_msg(dev,
                                    usb_sndctrlpipe(dev, 0),
                                    FTDI_SIO_SET_FLOW_CTRL_REQUEST,
                                    FTDI_SIO_SET_FLOW_CTRL_REQUEST_TYPE,
```

```
                                        0, priv->interface,
                                        buf, 0, WDR_TIMEOUT) < 0) {
                                err("urb failed to clear flow control");
                        }
                }

        }
        return;
} /* ftdi_termios */


static int ftdi_tiocmget (struct usb_serial_port *port, struct file *file)
{
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        unsigned char buf[2];
        int ret;

        dbg("%s TIOCMGET", __FUNCTION__);
        switch (priv->chip_type) {
        case SIO:
                /* Request the status from the device */
                if ((ret = usb_control_msg(port->serial->dev,
                                        usb_rcvctrlpipe(port->serial->dev, 0),
                                        FTDI_SIO_GET_MODEM_STATUS_REQUEST,
                                        FTDI_SIO_GET_MODEM_STATUS_REQUEST_TYPE,
                                        0, 0,
                                        buf, 1, WDR_TIMEOUT)) < 0 ) {
                        err("%s Could not get modem status of device - err: %d", __FUNCTION__,
                            ret);
                        return(ret);
                }
                break;
        case FT8U232AM:
        case FT232BM:
        case FT2232C:
                /* the 8U232AM returns a two byte value (the sio is a 1 byte value) - in the same
                   format as the data returned from the in point */
                if ((ret = usb_control_msg(port->serial->dev,
                                        usb_rcvctrlpipe(port->serial->dev, 0),
                                        FTDI_SIO_GET_MODEM_STATUS_REQUEST,
                                        FTDI_SIO_GET_MODEM_STATUS_REQUEST_TYPE,
                                        0, priv->interface,
                                        buf, 2, WDR_TIMEOUT)) < 0 ) {
                        err("%s Could not get modem status of device - err: %d", __FUNCTION__,
                            ret);
                        return(ret);
                }
                break;
        default:
                return -EFAULT;
                break;
        }

        return  (buf[0] & FTDI_SIO_DSR_MASK ? TIOCM_DSR : 0) |
                (buf[0] & FTDI_SIO_CTS_MASK ? TIOCM_CTS : 0) |
                (buf[0]  & FTDI_SIO_RI_MASK  ? TIOCM_RI  : 0) |
                (buf[0]  & FTDI_SIO_RLSD_MASK ? TIOCM_CD  : 0) |
                priv->last_dtr_rts;
}

static int ftdi_tiocmset(struct usb_serial_port *port, struct file * file, unsigned int set, unsigned int clear)
{
        int ret;

        dbg("%s TIOCMSET set:%02X clear:%02X", __FUNCTION__, set, clear);
        if (set & TIOCM_DTR){
          dbg("%s setting TIOCM_DTR", __FUNCTION__);
                if ((ret = set_dtr(port, HIGH)) < 0) {
                        err("Urb to set DTR failed");
                        return(ret);
                }
        }
        if (set & TIOCM_RTS) {
          dbg("%s setting TIOCM_RTS", __FUNCTION__);
                if ((ret = set_rts(port, HIGH)) < 0){
                        err("Urb to set RTS failed");
                        return(ret);
                }
        }

        if (clear & TIOCM_DTR){
          dbg("%s clearing TIOCM_DTR", __FUNCTION__);
                if ((ret = set_dtr(port, LOW)) < 0){
                        err("Urb to unset DTR failed");
                        return(ret);
                }
        }
        if (clear & TIOCM_RTS) {
          dbg("%s clearing TIOCM_RTS", __FUNCTION__);
                if ((ret = set_rts(port, LOW)) < 0){
                        err("Urb to unset RTS failed");
                        return(ret);
```

```
                }
        }
        return(0);
}


static int ftdi_ioctl (struct usb_serial_port *port, struct file * file, unsigned int cmd, unsigned long arg)
{
        struct ftdi_private *priv = usb_get_serial_port_data(port);

        int  ret, mask;

        dbg("%s cmd 0x%04x", __FUNCTION__, cmd);

        /* Based on code from acm.c and others */
        switch (cmd) {

        case TIOCMBIS: /* turns on (Sets) the lines as specified by the mask */
                dbg("%s TIOCMBIS", __FUNCTION__);
                if (get_user(mask, (unsigned long __user *) arg))
                        return -EFAULT;
                if (mask & TIOCM_DTR){
                        if ((ret = set_dtr(port, HIGH)) < 0) {
                                err("Urb to set DTR failed");
                                return(ret);
                        }
                }
                if (mask & TIOCM_RTS) {
                        if ((ret = set_rts(port, HIGH)) < 0){
                                err("Urb to set RTS failed");
                                return(ret);
                        }
                }
                return(0);
                break;

        case TIOCMBIC: /* turns off (Clears) the lines as specified by the mask */
                dbg("%s TIOCMBIC", __FUNCTION__);
                if (get_user(mask, (unsigned long __user *) arg))
                        return -EFAULT;
                if (mask & TIOCM_DTR){
                        if ((ret = set_dtr(port, LOW)) < 0){
                                err("Urb to unset DTR failed");
                                return(ret);
                        }
                }
                if (mask & TIOCM_RTS) {
                        if ((ret = set_rts(port, LOW)) < 0){
                                err("Urb to unset RTS failed");
                                return(ret);
                        }
                }
                return(0);
                break;

                /*
                 * I had originally implemented TCSET{A,S}{,F,W} and
                 * TCGET{A,S} here separately, however when testing I
                 * found that the higher layers actually do the termios
                 * conversions themselves and pass the call onto
                 * ftdi_sio_set_termios.
                 *
                 */

        case TIOCGSERIAL: /* gets serial port data */
                return get_serial_info(port, (struct serial_struct __user *) arg);

        case TIOCSSERIAL: /* sets serial port data */
                return set_serial_info(port, (struct serial_struct __user *) arg);

        /*
         * Wait for any of the 4 modem inputs (DCD,RI,DSR,CTS) to change
         * - mask passed in arg for lines of interest
         *   (use |'ed TIOCM_RNG/DSR/CD/CTS for masking)
         * Caller should use TIOCGICOUNT to see which one it was.
         *
         * This code is borrowed from linux/drivers/char/serial.c
         */
        case TIOCMIWAIT:
                while (priv != NULL) {
                        interruptible_sleep_on(&priv->delta_msr_wait);
                        /* see if a signal did it */
                        if (signal_pending(current))
                                return -ERESTARTSYS;
                        else {
                                char diff = priv->diff_status;

                                if (diff == 0) {
                                        return -EIO; /* no change => error */
                                }

                                /* Consume all events */
```

```
                                priv->diff_status = 0;

                                /* Return 0 if caller wanted to know about these bits */
                                if ( ((arg & TIOCM_RNG) && (diff & FTDI_RS0_RI)) ||
                                     ((arg & TIOCM_DSR) && (diff & FTDI_RS0_DSR)) ||
                                     ((arg & TIOCM_CD)  && (diff & FTDI_RS0_RLSD)) ||
                                     ((arg & TIOCM_CTS) && (diff & FTDI_RS0_CTS)) ) {
                                        return 0;
                                }
                                /*
                                 * Otherwise caller can't care less about what happened,
                                 * and so we continue to wait for more events.
                                 */
                        }
                }
                return(0);
                break;
        default:
                break;

        }


        /* This is not necessarily an error - turns out the higher layers will do
         *  some ioctls itself (see comment above)
         */
        dbg("%s arg not supported - it was 0x%04x - check /usr/include/asm/ioctls.h", __FUNCTION__, cmd);

        return(-ENOIOCTLCMD);
} /* ftdi_ioctl */


static void ftdi_throttle (struct usb_serial_port *port)
{
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        unsigned long flags;

        dbg("%s - port %d", __FUNCTION__, port->number);

        spin_lock_irqsave(&priv->rx_lock, flags);
        priv->rx_flags |= THROTTLED;
        spin_unlock_irqrestore(&priv->rx_lock, flags);
}


static void ftdi_unthrottle (struct usb_serial_port *port)
{
        struct ftdi_private *priv = usb_get_serial_port_data(port);
        int actually_throttled;
        unsigned long flags;

        dbg("%s - port %d", __FUNCTION__, port->number);

        spin_lock_irqsave(&priv->rx_lock, flags);
        actually_throttled = priv->rx_flags & ACTUALLY_THROTTLED;
        priv->rx_flags &= ~(THROTTLED | ACTUALLY_THROTTLED);
        spin_unlock_irqrestore(&priv->rx_lock, flags);

        if (actually_throttled)
                schedule_work(&priv->rx_work);
}

static int __init ftdi_init (void)
{
        int retval;

        dbg("%s", __FUNCTION__);
        retval = usb_serial_register(&ftdi_SIO_device);
        if (retval)
                goto failed_SIO_register;
        retval = usb_serial_register(&ftdi_8U232AM_device);
        if (retval)
                goto failed_8U232AM_register;
        retval = usb_serial_register(&ftdi_FT232BM_device);
        if (retval)
                goto failed_FT232BM_register;
        retval = usb_serial_register(&ftdi_FT2232C_device);
        if (retval)
                goto failed_FT2232C_register;
        retval = usb_serial_register(&ftdi_USB_UIRT_device);
        if (retval)
                goto failed_USB_UIRT_register;
        retval = usb_serial_register(&ftdi_HE_TIRA1_device);
        if (retval)
                goto failed_HE_TIRA1_register;
        retval = usb_register(&ftdi_driver);
        if (retval)
                goto failed_usb_register;

        info(DRIVER_VERSION ":" DRIVER_DESC);
        return 0;
failed_usb_register:
```

```
        usb_serial_deregister(&ftdi_HE_TIRA1_device);
failed_HE_TIRA1_register:
        usb_serial_deregister(&ftdi_USB_UIRT_device);
failed_USB_UIRT_register:
        usb_serial_deregister(&ftdi_FT2232C_device);
failed_FT2232C_register:
        usb_serial_deregister(&ftdi_FT232BM_device);
failed_FT232BM_register:
        usb_serial_deregister(&ftdi_8U232AM_device);
failed_8U232AM_register:
        usb_serial_deregister(&ftdi_SIO_device);
failed_SIO_register:
        return retval;
}


static void __exit ftdi_exit (void)
{

        dbg("%s", __FUNCTION__);

        usb_deregister (&ftdi_driver);
        usb_serial_deregister (&ftdi_HE_TIRA1_device);
        usb_serial_deregister (&ftdi_USB_UIRT_device);
        usb_serial_deregister (&ftdi_FT2232C_device);
        usb_serial_deregister (&ftdi_FT232BM_device);
        usb_serial_deregister (&ftdi_8U232AM_device);
        usb_serial_deregister (&ftdi_SIO_device);

}


module_init(ftdi_init);
module_exit(ftdi_exit);

MODULE_AUTHOR( DRIVER_AUTHOR );
MODULE_DESCRIPTION( DRIVER_DESC );
MODULE_LICENSE("GPL");

module_param(debug, bool, S_IRUGO | S_IWUSR);
MODULE_PARM_DESC(debug, "Debug enabled or not");


/*
 * Definitions for the FTDI USB Single Port Serial Converter -
 * known as FTDI_SIO (Serial Input/Output application of the chipset)
 *
 * The example I have is known as the USC-1000 which is available from
 * http://www.dse.co.nz - cat no XH4214 It looks similar to this:
 * http://www.dansdata.com/usbser.htm but I can't be sure There are other
 * USC-1000s which don't look like my device though so beware!
 *
 * The device is based on the FTDI FT8U100AX chip. It has a DB25 on one side,
 * USB on the other.
 *
 * Thanx to FTDI (http://www.ftdi.co.uk) for so kindly providing details
 * of the protocol required to talk to the device and ongoing assistence
 * during development.
 *
 * Bill Ryder - bryder@sgi.com formerly of Silicon Graphics, Inc.- wrote the
 * FTDI_SIO implementation.
 *
 * Philipp Ghring - pg@futureware.at - added the Device ID of the USB relais
 * from Rudolf Gugler
 *
 */

#define FTDI_VID          0x0403  /* Vendor Id */
#define FTDI_SIO_PID      0x8372  /* Product Id SIO application of 8U100AX  */
#define FTDI_8U232AM_PID 0x6001 /* Similar device to SIO above */
#define FTDI_8U232AM_ALT_PID 0x6006 /* FTDI's alternate PID for above */
#define FTDI_8U2232C_PID 0x6010 /* Dual channel device */
#define FTDI_RELAIS_PID 0xFA10  /* Relais device from Rudolf Gugler */
#define FTDI_NF_RIC_VID 0x0DCD  /* Vendor Id */
#define FTDI_NF_RIC_PID 0x0001  /* Product Id */

/* www.irtrans.de device */
#define FTDI_IRTRANS_PID 0xFC60 /* Product Id */

/* www.crystalfontz.com devices - thanx for providing free devices for evaluation ! */
/* they use the ftdi chipset for the USB interface and the vendor id is the same */
#define FTDI_XF_632_PID 0xFC08  /* 632: 16x2 Character Display */
#define FTDI_XF_634_PID 0xFC09  /* 634: 20x4 Character Display */
#define FTDI_XF_547_PID 0xFC0A  /* 547: Two line Display */
#define FTDI_XF_633_PID 0xFC0B  /* 633: 16x2 Character Display with Keys */
#define FTDI_XF_631_PID 0xFC0C  /* 631: 20x2 Character Display */
#define FTDI_XF_635_PID 0xFC0D  /* 635: 20x4 Character Display */
#define FTDI_XF_640_PID 0xFC0E  /* 640: Two line Display */
#define FTDI_XF_642_PID 0xFC0F  /* 642: Two line Display */

/* Video Networks Limited / Homechoice in the UK use an ftdi-based device for their 1Mb */
/* broadband internet service.  The following PID is exhibited by the usb device supplied */
```

```
/* (the VID is the standard ftdi vid (FTDI_VID) */
#define FTDI_VNHCPCUSB_D_PID 0xfe38 /* Product Id */

/*
 * The following are the values for the Matrix Orbital LCD displays,
 * which are the FT232BM ( similar to the 8U232AM )
 */
#define FTDI_MTXORB_0_PID       0xFA00  /* Matrix Orbital Product Id */
#define FTDI_MTXORB_1_PID       0xFA01  /* Matrix Orbital Product Id */
#define FTDI_MTXORB_2_PID       0xFA02  /* Matrix Orbital Product Id */
#define FTDI_MTXORB_3_PID       0xFA03  /* Matrix Orbital Product Id */
#define FTDI_MTXORB_4_PID       0xFA04  /* Matrix Orbital Product Id */
#define FTDI_MTXORB_5_PID       0xFA05  /* Matrix Orbital Product Id */
#define FTDI_MTXORB_6_PID       0xFA06  /* Matrix Orbital Product Id */

/* Interbiometrics USB I/O Board */
/* Developed for Interbiometrics by Rudolf Gugler */
#define INTERBIOMETRICS_VID                 0x1209
#define INTERBIOMETRICS_IOBOARD_PID         0x1002
#define INTERBIOMETRICS_MINI_IOBOARD_PID 0x1006

/* MGW added ER1 support */
#define EVOLUTION_VID       0xDEEE
#define EVOLUTION_ER1_PID 0x0300


#define BLACKCAT_VID        0x0403
#define BLACKCAT_GM10_PID 0xFDF0

/*
 * The following are the values for the Perle Systems
 * UltraPort USB serial converters
 */
#define FTDI_PERLE_ULTRAPORT_PID 0xF0C0 /* Perle UltraPort Product Id */

/*
 * The following are the values for the Sealevel SeaLINK+ adapters.
 * (Original list sent by Tuan Hoang.  Ian Abbott renamed the macros and
 * removed some PIDs that don't seem to match any existing products.)
 */
#define SEALEVEL_VID            0x0c52  /* Sealevel Vendor ID */
#define SEALEVEL_2101_PID       0x2101  /* SeaLINK+232 (2101/2105) */
#define SEALEVEL_2102_PID       0x2102  /* SeaLINK+485 (2102) */
#define SEALEVEL_2103_PID       0x2103  /* SeaLINK+2321 (2103) */
#define SEALEVEL_2104_PID       0x2104  /* SeaLINK+4851 (2104) */
#define SEALEVEL_2201_1_PID     0x2211  /* SeaPORT+2/232 (2201) Port 1 */
#define SEALEVEL_2201_2_PID     0x2221  /* SeaPORT+2/232 (2201) Port 2 */
#define SEALEVEL_2202_1_PID     0x2212  /* SeaPORT+2/485 (2202) Port 1 */
#define SEALEVEL_2202_2_PID     0x2222  /* SeaPORT+2/485 (2202) Port 2 */
#define SEALEVEL_2203_1_PID     0x2213  /* SeaPORT+2 (2203) Port 1 */
#define SEALEVEL_2203_2_PID     0x2223  /* SeaPORT+2 (2203) Port 2 */
#define SEALEVEL_2401_1_PID     0x2411  /* SeaPORT+4/232 (2401) Port 1 */
#define SEALEVEL_2401_2_PID     0x2421  /* SeaPORT+4/232 (2401) Port 2 */
#define SEALEVEL_2401_3_PID     0x2431  /* SeaPORT+4/232 (2401) Port 3 */
#define SEALEVEL_2401_4_PID     0x2441  /* SeaPORT+4/232 (2401) Port 4 */
#define SEALEVEL_2402_1_PID     0x2412  /* SeaPORT+4/485 (2402) Port 1 */
#define SEALEVEL_2402_2_PID     0x2422  /* SeaPORT+4/485 (2402) Port 2 */
#define SEALEVEL_2402_3_PID     0x2432  /* SeaPORT+4/485 (2402) Port 3 */
#define SEALEVEL_2402_4_PID     0x2442  /* SeaPORT+4/485 (2402) Port 4 */
#define SEALEVEL_2403_1_PID     0x2413  /* SeaPORT+4 (2403) Port 1 */
#define SEALEVEL_2403_2_PID     0x2423  /* SeaPORT+4 (2403) Port 2 */
#define SEALEVEL_2403_3_PID     0x2433  /* SeaPORT+4 (2403) Port 3 */
#define SEALEVEL_2403_4_PID     0x2443  /* SeaPORT+4 (2403) Port 4 */
#define SEALEVEL_2801_1_PID     0X2811  /* SeaLINK+8/232 (2801) Port 1 */
#define SEALEVEL_2801_2_PID     0X2821  /* SeaLINK+8/232 (2801) Port 2 */
#define SEALEVEL_2801_3_PID     0X2831  /* SeaLINK+8/232 (2801) Port 3 */
#define SEALEVEL_2801_4_PID     0X2841  /* SeaLINK+8/232 (2801) Port 4 */
#define SEALEVEL_2801_5_PID     0X2851  /* SeaLINK+8/232 (2801) Port 5 */
#define SEALEVEL_2801_6_PID     0X2861  /* SeaLINK+8/232 (2801) Port 6 */
#define SEALEVEL_2801_7_PID     0X2871  /* SeaLINK+8/232 (2801) Port 7 */
#define SEALEVEL_2801_8_PID     0X2881  /* SeaLINK+8/232 (2801) Port 8 */
#define SEALEVEL_2802_1_PID     0X2812  /* SeaLINK+8/485 (2802) Port 1 */
#define SEALEVEL_2802_2_PID     0X2822  /* SeaLINK+8/485 (2802) Port 2 */
#define SEALEVEL_2802_3_PID     0X2832  /* SeaLINK+8/485 (2802) Port 3 */
#define SEALEVEL_2802_4_PID     0X2842  /* SeaLINK+8/485 (2802) Port 4 */
#define SEALEVEL_2802_5_PID     0X2852  /* SeaLINK+8/485 (2802) Port 5 */
#define SEALEVEL_2802_6_PID     0X2862  /* SeaLINK+8/485 (2802) Port 6 */
#define SEALEVEL_2802_7_PID     0X2872  /* SeaLINK+8/485 (2802) Port 7 */
#define SEALEVEL_2802_8_PID     0X2882  /* SeaLINK+8/485 (2802) Port 8 */
#define SEALEVEL_2803_1_PID     0X2813  /* SeaLINK+8 (2803) Port 1 */
#define SEALEVEL_2803_2_PID     0X2823  /* SeaLINK+8 (2803) Port 2 */
#define SEALEVEL_2803_3_PID     0X2833  /* SeaLINK+8 (2803) Port 3 */
#define SEALEVEL_2803_4_PID     0X2843  /* SeaLINK+8 (2803) Port 4 */
#define SEALEVEL_2803_5_PID     0X2853  /* SeaLINK+8 (2803) Port 5 */
#define SEALEVEL_2803_6_PID     0X2863  /* SeaLINK+8 (2803) Port 6 */
#define SEALEVEL_2803_7_PID     0X2873  /* SeaLINK+8 (2803) Port 7 */
#define SEALEVEL_2803_8_PID     0X2883  /* SeaLINK+8 (2803) Port 8 */

/*
 * DSS-20 Sync Station for Sony Ericsson P800
 */
```

```
#define FTDI_DSS20_PID          0xFC82

/*
 * Home Electronics (www.home-electro.com) USB gadgets
 */
#define FTDI_HE_TIRA1_PID       0xFA78  /* Tira-1 IR transceiver */

/* USB-UIRT - An infrared receiver and transmitter using the 8U232AM chip */
/* http://home.earthlink.net/~jrhees/USBUIRT/index.htm */
#define FTDI_USB_UIRT_PID       0xF850  /* Product Id */

/* ELV USB Module UO100 (PID sent by Stefan Frings) */
#define FTDI_ELV_UO100_PID      0xFB58  /* Product Id */
/* ELV USB Module UM100 (PID sent by Arnim Laeuger) */
#define FTDI_ELV_UM100_PID      0xFB5A  /* Product Id */

/*
 * Definitions for ID TECH (www.idt-net.com) devices
 */
#define IDTECH_VID              0x0ACD  /* ID TECH Vendor ID */
#define IDTECH_IDT1221U_PID     0x0300  /* IDT1221U USB to RS-232 adapter */

/*
 * Definitions for Omnidirectional Control Technology, Inc. devices
 */
#define OCT_VID                 0x0B39  /* OCT vendor ID */
/* Note: OCT US101 is also rebadged as Dick Smith Electronics (NZ) XH6381 */
/* Also rebadged as Dick Smith Electronics (Aus) XH6451 */
/* Also rebadged as SIIG Inc. model US2308 hardware version 1 */
#define OCT_US101_PID           0x0421  /* OCT US101 USB to RS-232 */

/* an infrared receiver for user access control with IR tags */
#define FTDI_PIEGROUP_PID       0xF208  /* Product Id */

/*
 * Protego product ids
 */
#define PROTEGO_SPECIAL_1       0xFC70  /* special/unknown device */
#define PROTEGO_R2X0            0xFC71  /* R200-USB TRNG unit (R210, R220, and R230) */
#define PROTEGO_SPECIAL_3       0xFC72  /* special/unknown device */
#define PROTEGO_SPECIAL_4       0xFC73  /* special/unknown device */

/*
 * Gude Analog- und Digitalsysteme GmbH
 */
#define FTDI_GUDEADS_E808_PID   0xE808
#define FTDI_GUDEADS_E809_PID   0xE809
#define FTDI_GUDEADS_E80A_PID   0xE80A
#define FTDI_GUDEADS_E80B_PID   0xE80B
#define FTDI_GUDEADS_E80C_PID   0xE80C
#define FTDI_GUDEADS_E80D_PID   0xE80D
#define FTDI_GUDEADS_E80E_PID   0xE80E
#define FTDI_GUDEADS_E80F_PID   0xE80F
#define FTDI_GUDEADS_E888_PID   0xE888  /* Expert ISDN Control USB */
#define FTDI_GUDEADS_E889_PID   0xE889  /* USB RS-232 OptoBridge */
#define FTDI_GUDEADS_E88A_PID   0xE88A
#define FTDI_GUDEADS_E88B_PID   0xE88B
#define FTDI_GUDEADS_E88C_PID   0xE88C
#define FTDI_GUDEADS_E88D_PID   0xE88D
#define FTDI_GUDEADS_E88E_PID   0xE88E
#define FTDI_GUDEADS_E88F_PID   0xE88F

/*
 * Linx Technologies product ids
 */
#define LINX_SDMUSBQSS_PID      0xF448  /* Linx SDM-USB-QS-S */
#define LINX_MASTERDEVEL2_PID   0xF449  /* Linx Master Development 2.0 */
#define LINX_FUTURE_0_PID       0xF44A  /* Linx future device */
#define LINX_FUTURE_1_PID       0xF44B  /* Linx future device */
#define LINX_FUTURE_2_PID       0xF44C  /* Linx future device */

/* CCS Inc. ICDU/ICDU40 product ID - the FT232BM is used in an in-circuit-debugger */
/* unit for PIC16's/PIC18's */
#define FTDI_CCSICDU20_0_PID    0xF9D0
#define FTDI_CCSICDU40_1_PID    0xF9D1

/* Inside Accesso contactless reader (http://www.insidefr.com) */
#define INSIDE_ACCESSO          0xFAD0

/*
 * Intrepid Control Systems (http://www.intrepidcs.com/) ValueCAN and NeoVI
 */
#define INTREPID_VID            0x093C
#define INTREPID_VALUECAN_PID   0x0601
#define INTREPID_NEOVI_PID      0x0701

/*
 * Falcom Wireless Communications GmbH
 */
#define FALCOM_VID              0x0F94  /* Vendor Id */
#define FALCOM_TWIST_PID        0x0001  /* Falcom Twist USB GPRS modem */
```

```
/*
 * SUUNTO product ids
 */
#define FTDI_SUUNTO_SPORTS_PID  0xF680  /* Suunto Sports instrument */

/*
 * Definitions for B&B Electronics products.
 */
#define BANDB_VID               0x0856  /* B&B Electronics Vendor ID */
#define BANDB_USOTL4_PID        0xAC01  /* USOTL4 Isolated RS-485 Converter */
#define BANDB_USTL4_PID         0xAC02  /* USTL4 RS-485 Converter */
#define BANDB_USO9ML2_PID       0xAC03  /* USO9ML2 Isolated RS-232 Converter */

/*
 * RM Michaelides CANview USB (http://www.rmcan.com)
 * CAN fieldbus interface adapter, added by port GmbH www.port.de)
 * Ian Abbott changed the macro names for consistency.
 */
#define FTDI_RM_CANVIEW_PID     0xfd60  /* Product Id */

/*
 * EVER Eco Pro UPS (http://www.ever.com.pl/)
 */

#define EVER_ECO_PRO_CDS        0xe520  /* RS-232 converter */

/*
 * 4N-GALAXY.DE PIDs for CAN-USB, USB-RS232, USB-RS422, USB-RS485,
 * USB-TTY activ, USB-TTY passiv.  Some PIDs are used by several devices
 * and I'm not entirely sure which are used by which.
 */
#define FTDI_4N_GALAXY_DE_0_PID 0x8372
#define FTDI_4N_GALAXY_DE_1_PID 0xF3C0
#define FTDI_4N_GALAXY_DE_2_PID 0xF3C1

/*
 * Mobility Electronics products.
 */
#define MOBILITY_VID                    0x1342
#define MOBILITY_USB_SERIAL_PID         0x0202  /* EasiDock USB 200 serial */

/*
 * Active Robots product ids.
 */
#define FTDI_ACTIVE_ROBOTS_PID  0xE548  /* USB comms board */

/* Commands */
#define FTDI_SIO_RESET          0 /* Reset the port */
#define FTDI_SIO_MODEM_CTRL     1 /* Set the modem control register */
#define FTDI_SIO_SET_FLOW_CTRL  2 /* Set flow control register */
#define FTDI_SIO_SET_BAUD_RATE  3 /* Set baud rate */
#define FTDI_SIO_SET_DATA       4 /* Set the data characteristics of the port */
#define FTDI_SIO_GET_MODEM_STATUS       5 /* Retrieve current value of modern status register */
#define FTDI_SIO_SET_EVENT_CHAR 6 /* Set the event character */
#define FTDI_SIO_SET_ERROR_CHAR 7 /* Set the error character */
#define FTDI_SIO_SET_LATENCY_TIMER      9 /* Set the latency timer */
#define FTDI_SIO_GET_LATENCY_TIMER      10 /* Get the latency timer */


/*
 *   BmRequestType:  1100 0000b
 *   bRequest:       FTDI_E2_READ
 *   wValue:         0
 *   wIndex:         Address of word to read
 *   wLength:        2
 *   Data:           Will return a word of data from E2Address
 *
 */

/* Port Identifier Table */
#define PIT_DEFAULT             0 /* SIOA */
#define PIT_SIOA                1 /* SIOA */
/* The device this driver is tested with one has only one port */
#define PIT_SIOB                2 /* SIOB */
#define PIT_PARALLEL            3 /* Parallel */

/* FTDI_SIO_RESET */
#define FTDI_SIO_RESET_REQUEST FTDI_SIO_RESET
#define FTDI_SIO_RESET_REQUEST_TYPE 0x40
#define FTDI_SIO_RESET_SIO 0
#define FTDI_SIO_RESET_PURGE_RX 1
#define FTDI_SIO_RESET_PURGE_TX 2

/*
 * BmRequestType:  0100 0000B
 * bRequest:       FTDI_SIO_RESET
 * wValue:         Control Value
 *                   0 = Reset SIO
 *                   1 = Purge RX buffer
 *                   2 = Purge TX buffer
 * wIndex:         Port
 * wLength:        0
```

```
 * Data:              None
 *
 * The Reset SIO command has this effect:
 *
 *    Sets flow control set to 'none'
 *    Event char = $0D
 *    Event trigger = disabled
 *    Purge RX buffer
 *    Purge TX buffer
 *    Clear DTR
 *    Clear RTS
 *    baud and data format not reset
 *
 * The Purge RX and TX buffer commands affect nothing except the buffers
 *
   */

/* FTDI_SIO_SET_BAUDRATE */
#define FTDI_SIO_SET_BAUDRATE_REQUEST_TYPE 0x40
#define FTDI_SIO_SET_BAUDRATE_REQUEST 3

/*
 * BmRequestType:  0100 0000B
 * bRequest:       FTDI_SIO_SET_BAUDRATE
 * wValue:         BaudDivisor value - see below
 * wIndex:         Port
 * wLength:        0
 * Data:           None
 * The BaudDivisor values are calculated as follows:
 * - BaseClock is either 12000000 or 48000000 depending on the device. FIXME: I wish
 *   I knew how to detect old chips to select proper base clock!
 * - BaudDivisor is a fixed point number encoded in a funny way.
 *   (--WRONG WAY OF THINKING--)
 *   BaudDivisor is a fixed point number encoded with following bit weighs:
 *   (-2)(-1)(13..0). It is a radical with a denominator of 4, so values
 *   end with 0.0 (00...), 0.25 (10...), 0.5 (01...), and 0.75 (11...).
 *   (--THE REALITY--)
 *   The both-bits-set has quite different meaning from 0.75 - the chip designers
 *   have decided it to mean 0.125 instead of 0.75.
 *   This info looked up in FTDI application note "FT8U232 DEVICES \ Data Rates
 *   and Flow Control Consideration for USB to RS232".
 * - BaudDivisor = (BaseClock / 16) / BaudRate, where the (=) operation should
 *   automagically re-encode the resulting value to take fractions into consideration.
 * As all values are integers, some bit twiddling is in order:
 *   BaudDivisor = (BaseClock / 16 / BaudRate) |
 *   (((BaseClock / 2 / BaudRate) & 4) ? 0x4000    // 0.5
 *    : ((BaseClock / 2 / BaudRate) & 2) ? 0x8000  // 0.25
 *    : ((BaseClock / 2 / BaudRate) & 1) ? 0xc000  // 0.125
 *    : 0)
 *
 * For the FT232BM, a 17th divisor bit was introduced to encode the multiples
 * of 0.125 missing from the FT8U232AM.  Bits 16 to 14 are coded as follows
 * (the first four codes are the same as for the FT8U232AM, where bit 16 is
 * always 0):
 *   000 - add .000 to divisor
 *   001 - add .500 to divisor
 *   010 - add .250 to divisor
 *   011 - add .125 to divisor
 *   100 - add .375 to divisor
 *   101 - add .625 to divisor
 *   110 - add .750 to divisor
 *   111 - add .875 to divisor
 * Bits 15 to 0 of the 17-bit divisor are placed in the urb value.  Bit 16 is
 * placed in bit 0 of the urb index.
 *
 * Note that there are a couple of special cases to support the highest baud
 * rates.  If the calculated divisor value is 1, this needs to be replaced with
 * 0.  Additionally for the FT232BM, if the calculated divisor value is 0x4001
 * (1.5), this needs to be replaced with 0x0001 (1) (but this divisor value is
 * not supported by the FT8U232AM).
 */

typedef enum {
        SIO = 1,
        FT8U232AM = 2,
        FT232BM = 3,
        FT2232C = 4,
} ftdi_chip_type_t;

typedef enum {
 ftdi_sio_b300 = 0,
 ftdi_sio_b600 = 1,
 ftdi_sio_b1200 = 2,
 ftdi_sio_b2400 = 3,
 ftdi_sio_b4800 = 4,
 ftdi_sio_b9600 = 5,
 ftdi_sio_b19200 = 6,
 ftdi_sio_b38400 = 7,
 ftdi_sio_b57600 = 8,
 ftdi_sio_b115200 = 9
} FTDI_SIO_baudrate_t ;
```

```
/*
 * The ftdi_8U232AM_xxMHz_byyy constants have been removed. The encoded divisor values
 * are calculated internally.
 */

#define FTDI_SIO_SET_DATA_REQUEST FTDI_SIO_SET_DATA
#define FTDI_SIO_SET_DATA_REQUEST_TYPE 0x40
#define FTDI_SIO_SET_DATA_PARITY_NONE (0x0 << 8 )
#define FTDI_SIO_SET_DATA_PARITY_ODD (0x1 << 8 )
#define FTDI_SIO_SET_DATA_PARITY_EVEN (0x2 << 8 )
#define FTDI_SIO_SET_DATA_PARITY_MARK (0x3 << 8 )
#define FTDI_SIO_SET_DATA_PARITY_SPACE (0x4 << 8 )
#define FTDI_SIO_SET_DATA_STOP_BITS_1 (0x0 << 11 )
#define FTDI_SIO_SET_DATA_STOP_BITS_15 (0x1 << 11 )
#define FTDI_SIO_SET_DATA_STOP_BITS_2 (0x2 << 11 )
#define FTDI_SIO_SET_BREAK (0x1 << 14)
/* FTDI_SIO_SET_DATA */

/*
 * BmRequestType:  0100 0000B
 * bRequest:       FTDI_SIO_SET_DATA
 * wValue:         Data characteristics (see below)
 * wIndex:         Port
 * wLength:        0
 * Data:           No
 *
 * Data characteristics
 *
 *   B0..7   Number of data bits
 *   B8..10  Parity
 *           0 = None
 *           1 = Odd
 *           2 = Even
 *           3 = Mark
 *           4 = Space
 *   B11..13 Stop Bits
 *           0 = 1
 *           1 = 1.5
 *           2 = 2
 *   B14
 *           1 = TX ON (break)
 *           0 = TX OFF (normal state)
 *   B15 Reserved
 *
 */



/* FTDI_SIO_MODEM_CTRL */
#define FTDI_SIO_SET_MODEM_CTRL_REQUEST_TYPE 0x40
#define FTDI_SIO_SET_MODEM_CTRL_REQUEST FTDI_SIO_MODEM_CTRL

/*
 * BmRequestType:  0100 0000B
 * bRequest:       FTDI_SIO_MODEM_CTRL
 * wValue:         ControlValue (see below)
 * wIndex:         Port
 * wLength:        0
 * Data:           None
 *
 * NOTE: If the device is in RTS/CTS flow control, the RTS set by this
 * command will be IGNORED without an error being returned
 * Also - you can not set DTR and RTS with one control message
 */

#define FTDI_SIO_SET_DTR_MASK 0x1
#define FTDI_SIO_SET_DTR_HIGH ( 1 | ( FTDI_SIO_SET_DTR_MASK  << 8))
#define FTDI_SIO_SET_DTR_LOW  ( 0 | ( FTDI_SIO_SET_DTR_MASK  << 8))
#define FTDI_SIO_SET_RTS_MASK 0x2
#define FTDI_SIO_SET_RTS_HIGH ( 2 | ( FTDI_SIO_SET_RTS_MASK << 8 ))
#define FTDI_SIO_SET_RTS_LOW ( 0 | ( FTDI_SIO_SET_RTS_MASK << 8 ))

/*
 * ControlValue
 * B0    DTR state
 *          0 = reset
 *          1 = set
 * B1    RTS state
 *          0 = reset
 *          1 = set
 * B2..7 Reserved
 * B8    DTR state enable
 *          0 = ignore
 *          1 = use DTR state
 * B9    RTS state enable
 *          0 = ignore
 *          1 = use RTS state
 * B10..15 Reserved
 */

/* FTDI_SIO_SET_FLOW_CTRL */
#define FTDI_SIO_SET_FLOW_CTRL_REQUEST_TYPE 0x40
```

```
#define FTDI_SIO_SET_FLOW_CTRL_REQUEST FTDI_SIO_SET_FLOW_CTRL
#define FTDI_SIO_DISABLE_FLOW_CTRL 0x0
#define FTDI_SIO_RTS_CTS_HS (0x1 << 8)
#define FTDI_SIO_DTR_DSR_HS (0x2 << 8)
#define FTDI_SIO_XON_XOFF_HS (0x4 << 8)
/*
 *   BmRequestType:  0100 0000b
 *   bRequest:       FTDI_SIO_SET_FLOW_CTRL
 *   wValue:         Xoff/Xon
 *   wIndex:         Protocol/Port - hIndex is protocl / lIndex is port
 *   wLength:        0
 *   Data:           None
 *
 * hIndex protocol is:
 *   B0 Output handshaking using RTS/CTS
 *       0 = disabled
 *       1 = enabled
 *   B1 Output handshaking using DTR/DSR
 *       0 = disabled
 *       1 = enabled
 *   B2 Xon/Xoff handshaking
 *       0 = disabled
 *       1 = enabled
 *
 * A value of zero in the hIndex field disables handshaking
 *
 * If Xon/Xoff handshaking is specified, the hValue field should contain the XOFF character
 * and the lValue field contains the XON character.
 */

/*
 * FTDI_SIO_GET_LATENCY_TIMER
 *
 * Set the timeout interval. The FTDI collects data from the slave
 * device, transmitting it to the host when either A) 62 bytes are
 * received, or B) the timeout interval has elapsed and the buffer
 * contains at least 1 byte.  Setting this value to a small number
 * can dramatically improve performance for applications which send
 * small packets, since the default value is 16ms.
 */
#define FTDI_SIO_GET_LATENCY_TIMER_REQUEST FTDI_SIO_GET_LATENCY_TIMER
#define FTDI_SIO_GET_LATENCY_TIMER_REQUEST_TYPE 0xC0

/*
 *  BmRequestType:  1100 0000b
 *  bRequest:       FTDI_SIO_GET_LATENCY_TIMER
 *  wValue:         0
 *  wIndex:         Port
 *  wLength:        0
 *  Data:           latency (on return)
 */

/*
 * FTDI_SIO_SET_LATENCY_TIMER
 *
 * Set the timeout interval. The FTDI collects data from the slave
 * device, transmitting it to the host when either A) 62 bytes are
 * received, or B) the timeout interval has elapsed and the buffer
 * contains at least 1 byte.  Setting this value to a small number
 * can dramatically improve performance for applications which send
 * small packets, since the default value is 16ms.
 */
#define FTDI_SIO_SET_LATENCY_TIMER_REQUEST FTDI_SIO_SET_LATENCY_TIMER
#define FTDI_SIO_SET_LATENCY_TIMER_REQUEST_TYPE 0x40

/*
 *  BmRequestType:  0100 0000b
 *  bRequest:       FTDI_SIO_SET_LATENCY_TIMER
 *  wValue:         Latency (milliseconds)
 *  wIndex:         Port
 *  wLength:        0
 *  Data:           None
 *
 * wValue:
 *   B0..7   Latency timer
 *   B8..15  0
 *
 */

/*
 * FTDI_SIO_SET_EVENT_CHAR
 *
 * Set the special event character for the specified communications port.
 * If the device sees this character it will immediately return the
 * data read so far - rather than wait 40ms or until 62 bytes are read
 * which is what normally happens.
 */


#define FTDI_SIO_SET_EVENT_CHAR_REQUEST FTDI_SIO_SET_EVENT_CHAR
#define FTDI_SIO_SET_EVENT_CHAR_REQUEST_TYPE 0x40
```

```
/*
 *   BmRequestType:    0100 0000b
 *   bRequest:         FTDI_SIO_SET_EVENT_CHAR
 *   wValue:           EventChar
 *   wIndex:           Port
 *   wLength:          0
 *   Data:             None
 *
 * wValue:
 *   B0..7    Event Character
 *   B8       Event Character Processing
 *              0 = disabled
 *              1 = enabled
 *   B9..15   Reserved
 *
 */

/* FTDI_SIO_SET_ERROR_CHAR */

/* Set the parity error replacement character for the specified communications port */

/*
 *   BmRequestType:  0100 0000b
 *   bRequest:       FTDI_SIO_SET_EVENT_CHAR
 *   wValue:         Error Char
 *   wIndex:         Port
 *   wLength:        0
 *   Data:           None
 *
 *Error Char
 *   B0..7   Error Character
 *   B8      Error Character Processing
 *             0 = disabled
 *             1 = enabled
 *   B9..15  Reserved
 *
 */

/* FTDI_SIO_GET_MODEM_STATUS */
/* Retreive the current value of the modem status register */

#define FTDI_SIO_GET_MODEM_STATUS_REQUEST_TYPE 0xc0
#define FTDI_SIO_GET_MODEM_STATUS_REQUEST FTDI_SIO_GET_MODEM_STATUS
#define FTDI_SIO_CTS_MASK 0x10
#define FTDI_SIO_DSR_MASK 0x20
#define FTDI_SIO_RI_MASK  0x40
#define FTDI_SIO_RLSD_MASK 0x80
/*
 *    BmRequestType:    1100 0000b
 *    bRequest:         FTDI_SIO_GET_MODEM_STATUS
 *    wValue:           zero
 *    wIndex:           Port
 *    wLength:          1
 *    Data:             Status
 *
 * One byte of data is returned
 * B0..3 0
 * B4     CTS
 *          0 = inactive
 *          1 = active
 * B5     DSR
 *          0 = inactive
 *          1 = active
 * B6     Ring Indicator (RI)
 *          0 = inactive
 *          1 = active
 * B7     Receive Line Signal Detect (RLSD)
 *          0 = inactive
 *          1 = active
 */



/* Descriptors returned by the device
 *
 *  Device Descriptor
 *
 * Offset         Field            Size    Value    Description
 * 0     bLength          1       0x12     Size of descriptor in bytes
 * 1     bDescriptorType  1       0x01     DEVICE Descriptor Type
 * 2     bcdUSB           2       0x0110   USB Spec Release Number
 * 4     bDeviceClass     1       0x00     Class Code
 * 5     bDeviceSubClass  1       0x00     SubClass Code
 * 6     bDeviceProtocol  1       0x00     Protocol Code
 * 7     bMaxPacketSize0  1       0x08     Maximum packet size for endpoint 0
 * 8     idVendor         2       0x0403   Vendor ID
 * 10    idProduct        2       0x8372   Product ID (FTDI_SIO_PID)
 * 12    bcdDevice        2       0x0001   Device release number
 * 14    iManufacturer    1       0x01     Index of man. string desc
 * 15    iProduct         1       0x02     Index of prod string desc
 * 16    iSerialNumber    1       0x02     Index of serial nmr string desc
```

```
 * 17   bNumConfigurations 1    0x01    Number of possible configurations
 *
 * Configuration Descriptor
 *
 * Offset       Field                   Size    Value
 * 0    bLength                 1       0x09    Size of descriptor in bytes
 * 1    bDescriptorType         1       0x02    CONFIGURATION Descriptor Type
 * 2    wTotalLength            2       0x0020  Total length of data
 * 4    bNumInterfaces          1       0x01    Number of interfaces supported
 * 5    bConfigurationValue     1       0x01    Argument for SetCOnfiguration() req
 * 6    iConfiguration          1       0x02    Index of config string descriptor
 * 7    bmAttributes            1       0x20    Config characteristics Remote Wakeup
 * 8    MaxPower                1       0x1E    Max power consumption
 *
 * Interface Descriptor
 *
 * Offset       Field                   Size    Value
 * 0    bLength                 1       0x09    Size of descriptor in bytes
 * 1    bDescriptorType         1       0x04    INTERFACE Descriptor Type
 * 2    bInterfaceNumber        1       0x00    Number of interface
 * 3    bAlternateSetting       1       0x00    Value used to select alternate
 * 4    bNumEndpoints           1       0x02    Number of endpoints
 * 5    bInterfaceClass         1       0xFF    Class Code
 * 6    bInterfaceSubClass      1       0xFF    Subclass Code
 * 7    bInterfaceProtocol      1       0xFF    Protocol Code
 * 8    iInterface              1       0x02    Index of interface string description
 *
 * IN Endpoint Descriptor
 *
 * Offset       Field                   Size    Value
 * 0    bLength                 1       0x07    Size of descriptor in bytes
 * 1    bDescriptorType         1       0x05    ENDPOINT descriptor type
 * 2    bEndpointAddress        1       0x82    Address of endpoint
 * 3    bmAttributes            1       0x02    Endpoint attributes - Bulk
 * 4    bNumEndpoints           2       0x0040  maximum packet size
 * 5    bInterval               1       0x00    Interval for polling endpoint
 *
 * OUT Endpoint Descriptor
 *
 * Offset       Field                   Size    Value
 * 0    bLength                 1       0x07    Size of descriptor in bytes
 * 1    bDescriptorType         1       0x05    ENDPOINT descriptor type
 * 2    bEndpointAddress        1       0x02    Address of endpoint
 * 3    bmAttributes            1       0x02    Endpoint attributes - Bulk
 * 4    bNumEndpoints           2       0x0040  maximum packet size
 * 5    bInterval               1       0x00    Interval for polling endpoint
 *
 * DATA FORMAT
 *
 * IN Endpoint
 *
 * The device reserves the first two bytes of data on this endpoint to contain the current
 * values of the modem and line status registers. In the absence of data, the device
 * generates a message consisting of these two status bytes every 40 ms
 *
 * Byte 0: Modem Status
 *
 * Offset       Description
 * B0   Reserved - must be 1
 * B1   Reserved - must be 0
 * B2   Reserved - must be 0
 * B3   Reserved - must be 0
 * B4   Clear to Send (CTS)
 * B5   Data Set Ready (DSR)
 * B6   Ring Indicator (RI)
 * B7   Receive Line Signal Detect (RLSD)
 *
 * Byte 1: Line Status
 *
 * Offset       Description
 * B0   Data Ready (DR)
 * B1   Overrun Error (OE)
 * B2   Parity Error (PE)
 * B3   Framing Error (FE)
 * B4   Break Interrupt (BI)
 * B5   Transmitter Holding Register (THRE)
 * B6   Transmitter Empty (TEMT)
 * B7   Error in RCVR FIFO
 *
 */
#define FTDI_RS0_CTS    (1 << 4)
#define FTDI_RS0_DSR    (1 << 5)
#define FTDI_RS0_RI     (1 << 6)
#define FTDI_RS0_RLSD   (1 << 7)

#define FTDI_RS_DR  1
#define FTDI_RS_OE (1<<1)
#define FTDI_RS_PE (1<<2)
#define FTDI_RS_FE (1<<3)
#define FTDI_RS_BI (1<<4)
#define FTDI_RS_THRE (1<<5)
#define FTDI_RS_TEMT (1<<6)
```

```
#define FTDI_RS_FIFO   (1<<7)

/*
 * OUT Endpoint
 *
 * This device reserves the first bytes of data on this endpoint contain the length
 * and port identifier of the message. For the FTDI USB Serial converter the port
 * identifier is always 1.
 *
 * Byte 0: Line Status
 *
 * Offset       Description
 * B0    Reserved - must be 1
 * B1    Reserved - must be 0
 * B2..7        Length of message - (not including Byte 0)
 *
 */
```

REFERENCES

[1] Lars Cremean, William Dunbar, Dave Van Gogh, Jason Hickey, Eric Klavins, Jason Meltzer and Richard Murray, "The Caltech Multi-Vehicle Wireless Testbed", *in Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, NA, 2002, pp. TuA03-4.

[2] Raffaello D'Andrea and Michael Babish, "The RoboFlag Testbed", *in Proceedings of the American Control Conference*, Denver, CO, 2003, pp. 656-660.

[3] Ellis King, Yoshi Kuwata, Mehdi Alighanbari, Luca Bertuccelli, and Johnathan How, "Coordination and Control Experiments on a Multi-vehicle Testbed", *in Proceeding of the 2004 American Control Conference*, Boston, MA, 2004, pp. 5315-5320.

[4] Rafael Fierro and EzzAldeen Edwan, "The OSU Multi-vehicle Coordination Testbed", *in 45th Midwest Symposium on Circuits and Systems*, vol. 3, 2002, pp. III-41-III-44.

[5] Gabe Hoffmann, Dev Gorur Rajnarayan, Steven Waslander, David Dostal, Jung Soon Jang, and Claire Tomlin, "The Stanford Testbed of Autonomous Rotorcraft for Multi Agent Control (STARMAC)", *in 23rd Digital Avionics Systems Conference*, vol. 2, 2004, pp. 12.E.4-1.

[6] Mario Valenti, Brett Bethke, Gaston Fiore, Jonathan How, and Eric Feron,"Indoor Multi-Vehicle Flight Testbed for Fault Detection, Isolation, and Recovery", *in AIAA Guidance, Navigation, and Control Conference*, Keystone, CO, 2006, pp. 6200-6218.

[7] Vladimeros Vladimerou, Andrew Stubbs, Joel Rubel, Adam Fulford, Jeffrey Strick, Geir Dullerud, "A Hovercraft Testbed for Decentralized and Cooperative Control", *in Proceeding of the 2004 American Control Conference*, Boston, MA, 2004, pp. 1236-1241

[8] Adrian Ilie and Greg Welch, "Ensuring Color Consistency Across Multiple Cameras", *in Proceedings of the Tenth IEEE International Conference on Computer Vision*, 2005.

[9] James Bruce, Tucker Balch, and Manuela Veloso, "Fast and Cheap Color Image Segmentation for Interactive Robots", *School of Computer Science, Carnegie Mellon University*, Pittsburgh, PA, 2000.

[10] Richard Duda, Peter Hart, David Stork. 2001. *Pattern Classification.* John Wiley & Sons.

[11] Rafael Gonzalez, Richard Woods, Steven Eddins. 2004. *Digital Image Processing.* Pearson Prentice Hall.

[12] 68-95-99.7 Rule. *Internet Source*, 1 November 2006, available from http://en.wikipedia.org/wiki/Standard_deviation; accessed 1 November 2006.

[13] CCTV Factory. *Internet Source*, 3 May 2006, available from http://www.cctvfactory.com; accessed 3 May 2006.

[14] Linux Media Labs. *Internet Source*, 3 May 2006, available from http://www.linuxmedialabs.com; accessed 3 May 2006.

[15] Evolution Robotics. *Internet Source*, 15 January 2006, available from http://www.evolution.com/er1; accessed 15 January 2006.

[16] The Player/Stage Project. *Internet Source*, 2 November 2005, available http://playerstage.sourceforge.net/; accessed 2 November 2005.

[17] Roland Siegwart, Illah R. Nourbakhsh. 2004. *Introduction to Autonomous Mobile Robots.* The MIT Press.

[18] Sordalen, O.J. and DeWit, C. Canudas. 1992. "Exponential Control Law for a Mobile Robot: Extension to Path Following". *Proceedings of the International Control Conference on Robotics and Automation, Nice, France*, pp. 2158-2163.

[19] DeSantis, R.M.. 1995. "Path-Tracking for Car-Like Robots with Single and Double Steering". *IEEE Transactions on Vehicular Technology*. 44:366-377.

[20] L. Ljung, *System Identification: Theory for the User*, Prentice Hall, 1987.

[21] R. S. Sanchez Pena and M. Sznaier, *Robust Systems Theory and Applications*, John Wiley, New Jersey, 1998.

[22] T. Inanc, M. Sznaier, P. A. Parrilo and R. S. Sanchez Pena, "*Robust Identification with Mixed Parametric/Nonparametric Models and Time/Frequency-Domain Experiments: Theory and an Application*", IEEE Transactions on Control Systems Technology, July 2001, Vol. 9, No. 4, pp 608-617.

VITA

Travis Alan Riggs was born in Louisville, KY. Raised in neighboring Oldham County, Travis attended the county schools, graduating third in his class at Oldham County Senior High. With the option of attending the United States Naval Academy and Georgia Institute of Technology, Travis chose the University of Louisville as his home for undergraduate studies in Electrical Engineering. He completed his B.S. in 2005 and hopes to complete his M.Eng. in 2006. He now resides in Louisville with his lovely wife, Jaime.