Omnidirectional Vision System for a Soccer Playing Robot

Ben Abels

Mike Akers

Glenn Harden

Follow this and additional works at: http://opus.ipfw.edu/etcs_seniorproj_engineering

Part of the Engineering Commons

Opus Citation

Project Title: Omnidirectional Vision System for a Soccer Playing Robot

Team Members: Ben Abels (ECE)  
Mike Akers (CmpE)  
Glenn Harden (CmpE, ECE)

Advisor: Dr. Yanfei Liu (ECE)

Date: 5 May 2010
# Table of Contents

0. Table of Contents .......................................................................................................................... 2

1. Acknowledgments .......................................................................................................................... 4

2. Abstract .......................................................................................................................................... 5

3. Detailed Description of Conceptual Design .................................................................................. 6

   3.1 Overall Primary Design ............................................................................................................. 6

   3.2 Processor Design ....................................................................................................................... 6

       3.2.1 Motherboard ....................................................................................................................... 7

       3.2.2 Processor .......................................................................................................................... 8

       3.2.3 RAM .................................................................................................................................. 8

       3.2.4 Hard Disk Drive .................................................................................................................. 9

       3.2.5 Case .................................................................................................................................. 10

       3.2.6 Connection to Existing Microcontroller ............................................................................. 10

       3.2.7 Mounting to Existing Robot .............................................................................................. 10

   3.3 Power Supply ........................................................................................................................... 11

       3.3.1 Voltage Regulator .............................................................................................................. 11

       3.3.2 Power Source .................................................................................................................... 13

   3.4 Camera Design .......................................................................................................................... 14

   3.5 Mirror Design ........................................................................................................................... 16

       3.5.1 Mirror Choice and Overview ............................................................................................. 16

       3.5.2 Mirror Calculations ............................................................................................................ 17

       3.5.3 Mirror Backing Plate ......................................................................................................... 24

   3.6 Software Algorithm .................................................................................................................. 25

4. Omnidirectional Vision system – Build Process ......................................................................... 28

   4.1 Build Process Introduction ........................................................................................................ 28

   4.2 Processor Build ......................................................................................................................... 28

   4.3 Power Supply Build .................................................................................................................. 30

   4.4 Camera ..................................................................................................................................... 32

   4.5 Mirror ..................................................................................................................................... 33
1. Acknowledgments

We would like to thank our advisor, Dr Liu, for her advice and guidance. We also thank our sponsor, Raytheon, for our project funding as well as their recommendations. We would also like to thank IPFW for funding of the project.
2. Abstract

In August, 2007, Raytheon and IPFW initiated a 5-year project to develop a team of robots to compete in the RoboCup Middle Sized League. In the year of 2008-2009, an omnidirectional mobile robot platform was designed and built. The goal in the 2009-2010 year is to provide this robot the ability to “see”.

We are to create an omnidirectional vision system that is capable of locating and tracking an orange soccer ball anywhere on the playing field. The vision system we add to the robot must obey all rules and regulations of the RoboCup’s Middle Sized League. The vision system must also be able to be easily expanded in the future to allow necessary complex image processing in the competition.
3. Detailed Description of Conceptual Design

3.1 Overall Primary Design

The chosen design is the motherboard and processor, with a single camera pointed at a mirror. This design allows for the maximum possible processing power, as well as the highest number of available ports. However, this design also requires a power supply to be designed, or the current power supply to be modified to power the motherboard. The following figure, Figure 2, shows the interconnection between all of the components of the vision system with the existing system.

![Figure 1 - Diagram of Connection between Components](image-url)

3.2 Processor Design
3.2.1 Motherboard

The chosen motherboard for this project is the Mini-ITX motherboard from Intel, model DQ45EK. This motherboard has a standard LGA775 socket for the processor, supporting Core2Quad, Core2Duo, Pentium Dual-Core, Celeron Dual-Core, and Celeron processors that draw a maximum of 65W. It has two DIMM sockets for DDR2 RAM, and supports 800MHz and 667MHz RAM with up to a total of 4GB of RAM. It also has ten USB 2.0 ports, 4 internal SATA II ports, a serial port header, and a PCIe x1 connector.

![Motherboard Layout](image)

**Figure 2 - Motherboard Layout**

**Table 1 - Motherboard Connections**
### 3.2.2 Processor

The chosen processor is the Intel Core2Duo E8500. It is an LGA775 socket processor, which is what the motherboard requires. This processor is rated at 3.16GHz and a power consumption of 65W. It has a large L2 cache of 6MB, as opposed to the smaller 2MB, 3MB, or 4MB processors. This processor also comes with an included fan and heat sink to adequately cool the processor.

![Figure 3 - Intel E8500 Processor](image)

### 3.2.3 RAM

The motherboard supports 800MHz or 667MHz DDR2 RAM, and due to the high-speed processing that is required for this application, a speed of 800MHz was chosen. DDR2 800MHz RAM is called PC2 6400. Again, due to the high-speed nature of the project, dual-channel RAM was selected, as this speeds up the rate of data access and writing. Since the program and images will be relatively small, no more than 2GB of RAM will be required. 2GB of RAM in dual-channel mode means that there are two 1GB sticks of RAM that are required. The chosen RAM is G.Skill F2-6400CL4D-2GBPK.
3.2.4 Hard Disk Drive

Standard hard drives consist of read heads on an arm that travel across magnetic platters. The platters are constantly spinning, and moving the arm back and forth across these platters gives the read head access to each magnetic bit on the hard drive. Drive failures can occur when these read heads come in contact with the spinning platters, either demagnetizing a section of the platter or actually scratching the surface of the platter. The read head can come into contact with the platter when the hard drive feels an impact. Due to the possibility of impact between the robot and other objects (such as other robots), a standard hard drive is not a good option.

Figure 4 - Standard Hard Drive

An alternative to the standard hard drive is the Solid-State Drive (SSD). Solid-State Hard Drives do not have any moving parts, which increase their tolerance of physical shock.

Figure 5 - SATA Solid-State Hard Drive

The chosen hard drive is the Corsair CMFSSD-32N1 M32, which is a 32GB Solid-State Hard Drive. It has a SATA II connection, which is the same that the input to the motherboard has, and it is a 2.5" form factor.
hard drive, which is the width of the physical device. This SSD hard drive can sustain a 1500G shock and has a mean-time before failure of 1,500,000 hours.

3.2.5 Case

The chosen case is the Apex MW-100 Mini-ITX Case. This case is specifically designed for Mini-ITX motherboards, and has an internal 2.5” bay, which will be used for the SSD Hard Drive. This case will provide adequate cooling and protection to the motherboard and processor.

![Figure 6 - Apex MW-100 Mini-ITX Case](image)

3.2.6 Connection to Existing Microcontroller

The existing microcontroller will still be controlling the wheel motors and the kicking mechanism. Therefore, the new motherboard must communicate with the existing microcontroller to send direction and speed information to the existing microcontroller. This communication will be achieved by using a USB to Serial cable. The existing microcontroller has two serial ports that can be used for communication, while the new motherboard has multiple available USB ports.

3.2.7 Mounting to Existing Robot

The new motherboard and processor must be mounted to the existing robot, as the overall design of the robot will be kept constant. The motherboard, inside the case, will be mounted vertically between the compressed nitrogen tank and the existing microprocessor housing as shown in Figure 8.
3.3 Power Supply

The power supply section of the robot will be added in addition to the power supply that is currently there. The current power system will continue to run the existing microcontroller, the kicking mechanism, and the motor system. The new power supply will provide power to the hard drive and motherboard, which will provide power to the processor, RAM, and camera.

3.3.1 Voltage Regulator

The voltage regulator will need to be able to supply power to the hard drive and motherboard. The hard drive will require .48W when active. The motherboard will require approximately 50W, plus what is needed to power the processor, RAM, and camera. The processor will need 65W, the RAM needs 10W, and the camera will require 1.8W. Therefore the total power usage for the processor section is approximately 127W. The voltage regulator will therefore need to supply at least 127W in order to power the processor design. The voltage regulator will also need to supply all of the required voltages needed by the motherboard.
Table 2 - Main Power Connector Pinout

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal Name</th>
<th>Pin</th>
<th>Signal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+3.3 V</td>
<td>13</td>
<td>+3.3 V</td>
</tr>
<tr>
<td>2</td>
<td>+3.3 V</td>
<td>14</td>
<td>-12 V</td>
</tr>
<tr>
<td>3</td>
<td>Ground</td>
<td>15</td>
<td>Ground</td>
</tr>
<tr>
<td>4</td>
<td>+5 V</td>
<td>16</td>
<td>PS-ON# (power supply remote on/off)</td>
</tr>
<tr>
<td>5</td>
<td>Ground</td>
<td>17</td>
<td>Ground</td>
</tr>
<tr>
<td>6</td>
<td>+5 V</td>
<td>18</td>
<td>Ground</td>
</tr>
<tr>
<td>7</td>
<td>Ground</td>
<td>19</td>
<td>Ground</td>
</tr>
<tr>
<td>8</td>
<td>PWRGD (Power Good)</td>
<td>20</td>
<td>No connect</td>
</tr>
<tr>
<td>9</td>
<td>+5 V (Standby)</td>
<td>21</td>
<td>+5 V</td>
</tr>
<tr>
<td>10</td>
<td>+12 V</td>
<td>22</td>
<td>+5 V</td>
</tr>
<tr>
<td>11</td>
<td>+12 V (Note)</td>
<td>23</td>
<td>+5 V (Note)</td>
</tr>
<tr>
<td>12</td>
<td>2 x 12 connector detect (Note)</td>
<td>24</td>
<td>Ground (Note)</td>
</tr>
</tbody>
</table>

Note: When using a 2 x 10 power supply cable, this pin will be unconnected.

Table 3 - Processor Core Power Connector Pinout

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal Name</th>
<th>Pin</th>
<th>Signal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ground</td>
<td>2</td>
<td>Ground</td>
</tr>
<tr>
<td>3</td>
<td>+12 V</td>
<td>4</td>
<td>+12 V</td>
</tr>
</tbody>
</table>

To accomplish this task, we have chosen the M2-ATX Intelligent Vehicle DC-DC Power Supply available at www.mini-itx.com. The chosen voltage regulator is a DC to DC converter. This regulator is designed specifically to power Mini-ITX motherboards from a 12V DC source. It will provide a total of 160W of power, which is more than enough to power the motherboard, processor, camera, RAM, and hard drive. This regulator will also shut down power to the motherboard if the voltage supplied to the regulator is less than 11V for more than one minute. This prevents damage to the motherboard and processor due to under powering the equipment. The regulator will also handle voltages up to a maximum of 24 volts, with clamping occurring between 25V and 27V.
The voltage regulator provides 160W from a DC voltage of 12-24V, which means that the regulator will draw 13.4A of current. The power source for the voltage regulator will need to supply power for the entire duration of the game, which is 1 hour (two 15 minute halves with extra time for overtime and additional game delays).

The chosen battery is a 12V battery rated at 35Ah when discharged over a period of 20 hours. According to Peukert’s Law, $C_p = I^kt$, where $C_p$ is the capacity when discharged at a rate of 1A, $I$ is the discharge current, $k$ is a constant (1 for an ideal battery, 1.1-1.3 for a lead acid battery), and $t$ is the time of discharge. The constant $k$ used for these calculations is 1.3, since that is the worst-case scenario for a lead acid battery. Using Peukert’s Law, when this battery is discharged at a rate of 13.4A, the battery will last for approximately 1.42 hours.
\[ C_p = I^k t \]
\[ C_p = \left( \frac{35}{20} \right)^{1.3} t \]
\[ C_p = 41.4 \]
\[ 41.4 = 13.4^{1.3} t \]
\[ t = \frac{41.4}{13.4^{1.3}} \]
\[ t = 1.418 \]

The battery currently in the robot is a 25.9V Lithium Polymer Ion battery that is rated at 12.6Ah. Using these same calculations, the vision system alone would drain the existing battery in 0.376 hours, which is far short of the minimum time requirement for the robot.

3.4 Camera Design

The camera will be a single high-resolution camera. The chosen camera is a Mightex Systems BCE-C030-U color CMOS camera. It is an 8-bit 3 MP Color CMOS (complementary metal oxide semiconductor) Camera. With maximum resolution this camera can take 8 frames per second (FPS) at 2048x1536 resolution. At 1280x1024 this camera can take 24 FPS as well as a built-in 32 MB frame buffer. This allows the camera to hold 7 frames at full resolution (2048x1536). The camera also had built in filters for IR-cut (standard), and IR-pass if needed.

![Figure 9 - BCE-C030-U Enclosed Camera](image)

Including the C-mount (needed for the lens), and having the board enclosed. This camera is 58x58x39 mm in dimension and weighs 150g. The lens itself weighs an addition 72g and is Ø41.6mm x 48.8mm.
The power consumption of the camera is less than 1.8 W which is powered through the USB cable, which is within the range of the power supply. Since the camera has a USB digital output there is no need for a frame grabber. The only thing needed just an available USB port on the motherboard from the processor design section along with a compatible operating system (both Windows and non-Windows systems will work).
This system is sold with the BCE-C030-U camera, USB cable, an IR-cut filter (to avoid any un-needed Infrared signals the camera may pick up), and a CD (containing User’s Guide, SDK Manual, Windows-base application software with GUI, DirectShow driver, TWAIN Driver, and example codes in VB, C++, and Delphi).

We choose this camera over other cameras due to several restrictions we had to meet. We needed a small camera so that we may fit it on the existing robot and still fit inside the rules and regulations. We needed at least a 12080x1040 resolution camera so that the ball may be seen and recognized at long distances (22m at most). A frame rate of 25 FPS will allow us to achieve close to real time processing (humans see roughly at 24 FPS). USB 2.0 connections made the transfer between camera and processor fast and simple to use. The onboard frame buffer will help give us room for extra long image processes (such as first find of objects). The low power usage is beneficial since we are using onboard batteries. The lens is used to get a good focused image for the entire field.

### 3.5 Mirror Design

#### 3.5.1 Mirror Choice and Overview

The mirror is a Ginsberg Scientific 7-1301-K Spherical Convex Mirror shown in Figure 13. It will be suspended 2.75 cm above the camera. To achieve this we will be attaching the mirror to a thin sheet of aluminum to give a sturdy backing that can be attached to without interfering with the mirror surface. Having a backing on the mirror will limit the amount of light going to the camera. Ideally the only light getting to the camera would be from the mirror itself.
3.5.2 Mirror Calculations

The mirror had to be the right size and curvature to take in the entire field. To calculate this we had to look into many possibilities for the mirror setup. The field of view of the camera, the distance to the mirror, the width of the mirror, the amount of space the mirror must fit into, and the focal length of the mirror all play into the choice of the mirror.

The optical system for the robot does not need to look up. It needs a full view of the field but does not need to see the ceiling. Seeing out parallel to the top of the robot is the highest that the robot’s field of view needs to be. This would allow a full view of the field, the other robots, and the ball. The ball would be lost on high kicks but where it is going could be calculated, and this would allow the image to be more efficiently used. The vast majority of the time the ball will be in view half of the robots viewing field would not be wasted on the ceiling. A camera with twice our current resolution would be preferred if we were to also be viewing the ceiling.

The camera has a field of view of 67.0° - 35.3° in the vertical direction. This is the smallest direction so it is the constraining direction. We wanted the circle of the mirror to go right to the edge of the image in the vertical direction. This allows us to use the image most efficiently without special ordering an elliptical mirror. The outside of the dome mirror will show an image of what is directly out, parallel to the ground, from the robot. This means that a ray leaving the camera and contacting the outer edge of the mirror should leave parallel to the ground. This information allowed us to calculate the angle of the mirror at that location. In Figure 13 a line tangent to the mirror at the point the ray hits the mirror is drawn in.
The angle of the tangent line in Figure 14 can be found in terms of the angle $\theta$. We are considering the very small point where the ray hits the curved glass to be a flat surface. And when a ray hits a flat surface it at an angle $A$ it will leave at an angle $180 - A$. This is shown in Figure 14.

![Figure 13 - Tangent line to where the ray hits the curved mirror](image)

![Figure 14 - Reflection Angle](image)

Using this principle we were able to determine the angle $\alpha$ in the Figure 15.
Figure 15 includes all the needed labeled angles and lengths that will be discussed in the mirror calculations. The mirror is not going to be a full circle and this is represented by the circle fading to grey at one point. This is theoretically where the mirror could cut off; it could extend further if that is what is available. The Ginsberg Scientific 7-1301-K Spherical Convex Mirror does cut off within a few millimeters of where the ray at angle \( \theta \) will hit it.

Having the angle \( \alpha \) allows the calculation of the angle \( \phi \). The two combined make a 90° angle.

\[
\alpha = 45° + \frac{\theta}{2} \\
\phi = 45° - \frac{\theta}{2}
\]
All of the solutions for the curvature of the mirror and the size of the mirror must be found in terms of other variables. The variables used are \( \theta \) and \( h \), which is the distance from the camera to where the mirror would set if it were cut off at the points outside ray hits the mirror. \( \theta \) is chosen because it has a limited range from \( 38.5^\circ - 17.7^\circ \). The other variable \( h \) is chosen because it is limited because there are only 18cm above the robot that we can use. The camera and lens already stand at 8.8cm. This only leaves about 9cm as the maximum for the \( h \) value.

The minimum width of the mirror is given by \( w \), it is also the distance from the point the two furthest rays hit the mirror. It can be found in terms of \( h \), and \( \theta \) using a triangle with sides \( h \) and \( \frac{w}{2} \) and an angle \( \theta \) which is shown in Figure 16.

![Figure 16 - Triangle to calculate w](image)

\[
\tan(\theta) = \frac{w}{2h}
\]

\[
w = \tan(\theta) \times 2h
\]

The radius of the circle \( r \) is next found. A triangle with a side \( r \), side \( \frac{w}{2} \) and angle \( \phi \) can be found using Figure 15. This triangle is shown below in Figure 17. The radius can be found in term of \( w \) and \( \phi \) which have both been found in terms of \( \theta \) and \( h \) which are our two variables.
\[
\sin(\phi) = \frac{w}{2r}
\]

\[
r = \frac{w}{2 \sin(\phi)}
\]

\[
r = \frac{\tan(\theta) \times 2h}{2 \sin(\phi)}
\]

\[
r = \frac{\tan(\theta) \times 2h}{2 \sin\left(45^\circ - \frac{\theta}{2}\right)}
\]

The calculation to find \(d\), the distance from the front of the mirror to the camera is shown below. The Figure 18 shows the lines and triangles used to figure up the value for \(d\). It also is found in terms of \(h\) and \(\theta\).
Figure 18 - Image to calculate distance between camera and mirror

\[ r + d = h + r \cdot \cos(\phi) \]

\[ d = h + r \cdot (\cos(\phi) - 1) \]

\[ d = h + r \cdot \left(\cos\left(45^\circ - \frac{\theta}{2}\right) - 1\right) \]

\[ d = h + \frac{\tan(\theta) \cdot 2h \cdot \left(\cos\left(45^\circ - \frac{\theta}{2}\right) - 1\right)}{2 \sin\left(45^\circ - \frac{\theta}{2}\right)} \]

\[ d = h + \frac{\tan(\theta) \cdot h \cdot \left(\cos\left(45^\circ - \frac{\theta}{2}\right) - 1\right)}{\sin\left(45^\circ - \frac{\theta}{2}\right)} \]

This allows for a spreadsheet to be made with the two variables changing. The h stays the same in each table while the θ changes through its full range. Table 5 has h at 3 cm, Table 6 has h at 5 cm, and Table 7 has h at 8 cm.
Table 5 - Calculations of w and focal length with h at 3 cm

<table>
<thead>
<tr>
<th>θ (degree)</th>
<th>h (cm)</th>
<th>w (cm)</th>
<th>d (cm)</th>
<th>r (cm)</th>
<th>focal length</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>3</td>
<td>1.61</td>
<td>2.48</td>
<td>2.50</td>
<td>1.25</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>2.18</td>
<td>2.34</td>
<td>3.63</td>
<td>1.82</td>
</tr>
<tr>
<td>25</td>
<td>3</td>
<td>2.80</td>
<td>2.22</td>
<td>5.00</td>
<td>2.50</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>3.46</td>
<td>2.10</td>
<td>6.69</td>
<td>3.35</td>
</tr>
<tr>
<td>35</td>
<td>3</td>
<td>4.20</td>
<td>2.00</td>
<td>8.84</td>
<td>4.42</td>
</tr>
<tr>
<td>40</td>
<td>3</td>
<td>5.03</td>
<td>1.91</td>
<td>11.63</td>
<td>5.82</td>
</tr>
</tbody>
</table>

Table 6 - Calculations of w and focal length with h at 5 cm

<table>
<thead>
<tr>
<th>θ (degree)</th>
<th>h (cm)</th>
<th>w (cm)</th>
<th>d (cm)</th>
<th>r (cm)</th>
<th>focal length</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>5</td>
<td>2.68</td>
<td>4.14</td>
<td>4.17</td>
<td>2.08</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>3.64</td>
<td>3.91</td>
<td>6.05</td>
<td>3.03</td>
</tr>
<tr>
<td>25</td>
<td>5</td>
<td>4.66</td>
<td>3.70</td>
<td>8.33</td>
<td>4.17</td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td>5.77</td>
<td>3.51</td>
<td>11.15</td>
<td>5.58</td>
</tr>
<tr>
<td>35</td>
<td>5</td>
<td>7.00</td>
<td>3.34</td>
<td>14.73</td>
<td>7.36</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>8.39</td>
<td>3.18</td>
<td>19.38</td>
<td>9.69</td>
</tr>
</tbody>
</table>

Table 7 - Calculations of w and focal length with h at 8 cm

<table>
<thead>
<tr>
<th>θ (degree)</th>
<th>h (cm)</th>
<th>w (cm)</th>
<th>d (cm)</th>
<th>r (cm)</th>
<th>focal length</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>8</td>
<td>4.29</td>
<td>6.62</td>
<td>6.67</td>
<td>3.33</td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>5.82</td>
<td>6.25</td>
<td>9.68</td>
<td>4.84</td>
</tr>
<tr>
<td>25</td>
<td>8</td>
<td>7.46</td>
<td>5.91</td>
<td>13.33</td>
<td>6.67</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>9.24</td>
<td>5.61</td>
<td>17.85</td>
<td>8.92</td>
</tr>
<tr>
<td>35</td>
<td>8</td>
<td>11.20</td>
<td>5.34</td>
<td>23.57</td>
<td>11.78</td>
</tr>
<tr>
<td>40</td>
<td>8</td>
<td>13.43</td>
<td>5.09</td>
<td>31.01</td>
<td>15.51</td>
</tr>
</tbody>
</table>

This gave a range of possible mirror sizes and focal lengths. These were the two details given most often about the mirrors. The Ginsberg Scientific 7-1301-K Spherical Convex Mirror has a 5 cm width and a 5 cm focal length. With these specifications the mirror backing will have to be 4.00 cm away, which places the front of the mirror 2.75 cm away. The other values for the mirrors calculated position are seen in Table 8.
Table 8 - Calculated values for Ginsberg Scientific 7-1301-K Spherical Convex Mirror

<table>
<thead>
<tr>
<th>θ (degree)</th>
<th>h (cm)</th>
<th>w (cm)</th>
<th>d (cm)</th>
<th>r (cm)</th>
<th>focal length</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>4.00</td>
<td>5.00</td>
<td>2.75</td>
<td>9.98</td>
<td>4.99</td>
</tr>
</tbody>
</table>

3.5.3 Mirror Backing Plate

The mirror will be held 2.75 cm above the camera on an aluminum backing plate. This will be held up by four steel rods that come up from the original base; this is shown in Figure 19.

The aluminum backing plate will be the same kind as the aluminum used in the rest of the robot but it will be much thinner. The reason this aluminum was chosen by the 2008-2009 project team was due to its low weight and its sturdiness. This was figured up by last year’s senior design team who looked over different metal types. We will be using a thinner sheet of this same aluminum because we are not placing it in an area that will come into contact with other robots. It is also not holding up a substantial weight. It will also not be build on in future years.

The aluminum backing plate will be a circle with a 40.5 cm diameter. This will keep the size of the upper part of the robot smaller than the lower part. This allows the lower part of the robot to absorb impacts from other robots. The upper part will only have to be build strong enough to handle the ball hitting it and the robot shaking if it is hit.

To hold the aluminum plate at the 2.75 cm height above the camera we will use $\frac{3}{8}$ inch diameter steel rods. These were chosen over the aluminum because the aluminum would have to be thicker than the steel to perform the same job. Where the rods are located they will obstruct the view. So it is more important for these particular parts to be smaller rather than lighter. These beams will extend 28.5 cm
from where they are attached on the four main vertical supports of the robot. This will put the aluminum 4.00 cm above the camera. This leaves enough room that the mirror will be 2.75 cm above the camera when it is attached as shown in Figure 20. This puts the final height at 75 cm. This is within the 80 cm maximum height and leaves room for slight adjustment.

Figure 20 - Mirror rod design

3.6 Software Algorithm

The image processing will occur in several steps. The first step is to send the image through a RGB-filter so that the objects we are looking for will become “blobs” that stand apart from the background. We take the image and filter out any un-wanted colors and fill the rest in with black. Any pixels that fall into the predetermined range of orange will be turned to a single “true” orange. The same will be done for white, green and black (black will made a different color). This will allow us to work with our new image more easily without worrying about the background.

Figure 21 - RGB Filtering
Now that the image is filtered we can scan the image for the ball. We will scan the image every four pixels across and in an offset fashion for the following rows (a checkerboard pattern). This is done to speed up processing time as we do not need to scan every pixel for the ball (the ball will take multiple pixels). Once the ball is found (the color orange) then will proceed pixel by pixel in a circular motion until all the pixels have been found. This will be known as the entire ball will be found and outlined by the background color will filter to in the first step (in this case black). We will do this process for each object we are looking for, such as other robots and goal posts. Since the ball is now found we can add edge detection filtering as well as object tracking algorithms. Edge detection will reduce the amount of processing even farther because now we are only tracking the edge of the object instead of the entire object.

![Figure 22 - Edge Detection](image)

We will track the objects by predicting where the object will be in the next frame. First, two subsequent frames will be saved. The center of the object will be found in each frame. The distance and direction from the front of the camera will be found in each frame. Since we know the speed of the frames, 25 FPS, and we can determine by the angle of the mirror how far a certain amount of pixels are, from this we can determine the distance and direction the ball is moving.
\[ d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2} \]

\[ v = \frac{d}{\left(\frac{1}{FPS}\right)} \]

\[ X_3 = X_2 + (X_2 - X_1) \]

\[ Y_3 = Y_2 + (Y_2 - Y_1) \]

Figure 23 - Determining future position of an object

This will allow us to predict where the object will be next and scan only the area in a 10° range from the sides of the object. This additional step will allow the processing to be accomplished even faster.

Dependant on the distance to where the ball is we will use one of three possible robot movement algorithms. The first is if the object is far away. We assume the ball is going to continue moving at the same speed and same distance forever. We will then “plot” an intercept course to the objects final spot. The second algorithm is if the ball is close. We will then move to where the ball currently is. The reason we only use when the object is close is because if the ball is far out, the robot will end up “chasing” the object and never catch it. The third algorithm is literally the average of the first two. This is used when the object is mid-range from the robot.

From the calculations obtained from the object tracking, we will take this information and pass it through the serial port of the motherboard and send it to the existing micro-controller. This information will contain distance and direction. The existing micro-controller is already programmed to understand distance and direction and will determine the power that is needed to be applied to each motor.
4. Omnidirectional Vision system – Build Process

4.1 Build Process Introduction

The first step to our project was to order the parts needed to build our processor, camera, mirror and mounting supplies. We searched many websites, such as Tiger Direct, Fry's and Newegg, for the best prices for our processing supplies. We searched Lowe's, Menards, RadioShack and other local supply stores for parts to use in our omnidirectional vision system. We started this process at the beginning of the Spring 2010 semester. This allowed us enough time to prepare all final design modifications while the parts were being shipped.

With the complexity of the design build reaching across multiple sections we will report each section separately: we have the Processor Design, Power Design, Camera Design, Mirror Design, Software Design, Mounting Design, and Integration sections.

4.2 Processor Build

We start the processor design by first ordering our parts needed. We ordered an Intel Core 2 Duo E8500 3.16 GHz Processor and Intel Socket LGA 775 DQ45 EK Mini ITX Motherboard as our base for the processing system. We also ordered our case, memory (G.Skill F2-6400CL4D-2GBPK), and solid state hard drive. Below are some pictures of items that were ordered.
Figure 25 - Apex MW-100 Mini-ITX Case

Figure 26 Transcend 2.5" SATA-2 SSD TS6GSSD25S-S Solid State Hard Drive

Figure 27 - Dynatron Low Profile CPU Heat sink/Fan
During the ordering process we made a change to the original part list. The solid state hard drive we were going to order (Corsair CMFSSD-32N1 M32 Solid State) was no longer in stock. We found a smaller, faster and cheaper replacement solid state hard drive (Transcend 2.5" SATA-2 SSD TS8GSSD25S-S).

We then installed the motherboard into the case. Using care not to short out any electronics we carefully mounted the motherboard using the mounting screws and placement configuration provided. Next the processor was installed into the motherboard. Extra care needs to be taken to make sure that none of the pins on the underside of the processor chip are damaged when installed. The memory was installed next, 2 sticks of 1 GB memory in the dual channel array.

We then came upon our first problem of the build. As we installed the heat sink and fan to keep the processor cool and running properly we noticed that the fan’s height was too tall to allow the case lid to close. That is when we ordered the replacement heat sink/fan and thermal compound. The thermal compound was needed to help with the heat transfer between the top of the processor and the bottom of the heat sink. The next problem we had with the CPU fan was that the motherboard automatic speed control was not functionally properly. We found that the motherboard's BIOS was correctly reading the temperature of the CPU but was not increasing the voltage (and RPMs) of the fan. This would cause the CPU to always run at its lowest speed causing the CPU to overheat and shutdown. After trying several connections and fixes we decided to run the fans control voltage to always HIGH. This kept a constant 5V voltage on the fan and therefore running full speed all the time.

Lastly, we installed Linux (Ubuntu) as our operating system. Later on we found that this would be a problem as the drivers that came with the camera did not work properly on the Linux compiler. We received a copy of Windows XP from the IPFW Engineering Department and installed Visual Studio for our C++ programming.

### 4.3 Power Supply Build

We started the power supply process by ordering an Amstron 35Ah 12V battery. This would provide us with sufficient power to run for the entire competition. We then wired an enercell adaptaplug socket (size N) to provide power to the Apex case. We originally planned to use a M2-ATX Intelligent Vehicle DC-DC Power Supply to convert from the battery to the motherboard but we found that the Apex MW-100 Mini-ITX Case came with a DC-DC power supply that was able to withstand 11-13v.
We powered the processor with the Amstron 35 Ah battery until failure. The battery failed to power processor and camera after 7 hours of continuous run time, which was well above our requirement of 45 minutes.

We fully charged the battery to run the test again for verification. This is where we ran into our first power supply problem. The Apex Case's power supply has a safety built in that it will not power the motherboard if the source is producing more than 12.7V. Our battery was producing an average of 13.1V after a full charge. We found that if we drained the battery to under 12.7V the power supply would then allow power to the motherboard.

The next problem we ran into was that the Amstron 35Ah battery was physically too large, both in dimensions and weight. The battery weight 27 lb putting us over our weight requirement. Since the power requirement was only 45 minutes we decided on a smaller Werker WKA12-10F2 12V 10Ah AGM Sealed Non-Spillable battery.
This Werker 10Ah battery fit inside the robot and it weighed only 8lb. This fit into our dimension and weight requirements. We then tested the battery for run time. We ran into the same problem again with overcharging to 13V, but were able to discharge to 12.7V and run the processor and camera for the full length of the competition.

4.4 Camera

The camera design stayed the same as in the conceptual design. We used a Mightex brand BCE-C030-U color CMOS Camera. This is a 3 megapixel camera that sends the images over the USB interface. To focus the image to what we need we attached a Mightex H2Z0414C-MP Lens, both the camera and lens are c-mount compatible so this made and easy connection.

The biggest issue with the camera is that the Linux drivers provided by Mightex did not compile correctly as it require several header files that could not be acquired. As a result we installed Windows and used those drivers to successfully take pictures from the camera. The second biggest issue is that the camera
only produced 6-7 frames per second. This was well beneath our expected 30 frames per second, but it still accomplished the job tracking a dynamic orange soccer ball.

Figure 30 - Mightex BCE-C030-U Color CMOS Camera

4.5 Mirror

The mirror chosen from last semester was a Ginsberg Scientific 7-1301-K Spherical Convex Mirror. It was found to meet the minimum requirements to work with the chosen camera and to allow the robot to view the entire field. The downside of the mirror was that we had very little flexibility. It had to be very precisely located to allow us to see the entire field and if there were small difference between the theoretical numbers and what the camera actually saw then the mirror would have not been a success. Despite these short-comings it was the best that had been found at the time.

When a better idea for a mirror arrived, it was given a try. The reflective half of a spherical chrome top light bulb was used. It was temporarily taped to the robot to give it a test run and see if it allowed vision of the entire field as is shown in the picture below. It was more curved than the Ginsberg Scientific 7-1301-K Spherical Convex Mirror which makes it easier to see the entirety of the field. The reflective half
of the spherical chrome top light bulb was curved enough that small alterations in the location of the mirror or zoom of the camera did not cause the far points of the field to go out of sight.

Figure 31 - Robot with reflective half of a spherical chrome top light bulb temporarily attached

The only difficulty with the mirror itself was in finding mirrors with a small enough radius of curvature. Images of the final mounting of the mirror and discussion of how it was permanently mounted to the robot can be found in the mounting section.

4.6 Software

The software design has four main components:

1. Retrieve the image from the camera.
2. Process the image.
3. Display the image on the screen.
4. Communicate with the existing microcontroller.
To retrieve the image from the camera, the computer uses the APIs provided by the camera manufacturer, Mightex Systems. These APIs include functions to set the gains of the individual RGB components, set the resolution of the image, and set other device modes. The image is then stored in an array pointed to by a pointer called ‘location.’ This array has 56 bytes of header information, and then contains the RGB components for each pixel.

After the program has retrieved the image, it compares each pixel against its color thresholds for each of the four colors. If the pixel falls within one of the colors, it sets the corresponding element in a 2-dimensional array, called either ‘white,’ ‘black,’ ‘green,’ or ‘orange.’ It then goes through the algorithm to detect the orange ball using the ‘orange’ array. It calculates the distance in pixels from the center of the mirror to the ball, and then converts that to feet. It also calculates the angle between the front of the robot and the ball.

The keyboard is used to determine the actual image to be displayed. If the real image is selected to be displayed, the program creates a new array of every other pixel. If one of the blob processed images is selected to be displayed, the program uses the corresponding color array (which were defined in the previous step) and sets the pixel on a new array to be the chosen color (e.g. all pixels within the orange threshold will show up as a true orange in the new array). This new array is then displayed on the screen using the OpenGL Utility Toolkit (GLUT) library functions.

After the ball has been located, the ball’s location must be transferred to the microcontroller. The communication with the microcontroller has been defined to be a packet of three bytes. There are two bytes of information sent to the microcontroller, and one byte of information sent back from the microcontroller. There are five pieces of information that need to be communicated to the microcontroller:

- Angle to move: 6 bits
- Speed to move: 4 bits
- Speed to turn: 4 bits
- Allow microcontroller to kick when proximity sensor is activated: 1 bit
- Force microcontroller to kick immediately: 1 bit

The two bytes of information sent to the microcontroller are defined by:

**Byte 1:**
- The six most significant bits are the angle for the robot to move.
- The remaining two bits are the most significant bits of the movement speed.

**Byte 2:**
- The two most significant bits are the least significant bits of the movement speed.
- The next four bits are the speed for the robot to turn.
The next bit allows the microcontroller to kick when the proximity sensor is active. The last bit forces the microcontroller to kick immediately.

The microcontroller also has some information which is useful to the computer when actually playing soccer. The byte that is returned from the microcontroller is defined to be:

- **Bit 7**: Status of the program switch
- **Bit 6-3**: Not used
- **Bit 2**: Status of reed switch to tell if the kicking plate is extended
- **Bit 1**: Status of reed switch to tell if the kicking plate is retracted
- **Bit 0**: Status of proximity sensor

This communication is realized by using a serial port. The chosen serial port is a USB to serial connector, which is installed as COM2. The chosen baud rate is 19200 bps, with an 8-bit data byte, one stop bit, and no parity bit.

There are several primary functions that users need to use in order for the camera to work. The first of these is called `cameraInit()`. This function initializes all of the camera functions (i.e. setting RGB gain, resolution, and decimation level). The next primary function is `glutInit()`, which initializes all of the GLUT libraries and functions to allow for displaying graphics on the screen. `serialInit()` initializes the serial port for communication with the microcontroller. `glutKeyboardFunc(keyboard)` sets up the function `void keyboard(unsigned char key, int x, int y)` as the event handler for keyboard events, and `glutDisplayFunc(display)` sets the function `display()` as the function to handle repaint events. `display()` has two purposes. The first is to obtain the new image, and the second is to display the image on the screen. Obtaining the new image is handled by calling the function `readImage()`. `readImage()` gets the new image from the camera, performs the required blob processing (by calling the function `MakeBlobs()`), and finds the current ball position (by calling the function `BallLocation()`). The function then updates the microcontroller with the current direction and speed that the robot should move.

The code written by this year’s project is included in section 9. Computer Code of the report and the flow chart below provides a graphical representation of the image processing path.
4.7 Mounting

4.7.1 Camera Mounting

The camera was mounted in the location that was planned during the conceptual design. The robot rises up to a platform for the camera to rest on. The camera itself has 4 threaded screw holes, one on each side. These were used to attach brackets made from 22ga Weld Steel Sheets to the camera. It was decided that only three of the four would be used because the fourth would run into trouble with blocking the USB connection to the camera.

These brackets were bent and bolted down to the frame of the robot. Under the camera itself, weather stripping was used as a cushion as it had been by the group which built the base. The camera sitting up here could easily be hit by a ball, but when the mirror was attached, it protected the camera below it.

Figure 32 - Camera mounted
4.7.2 Mirror Mounting

The mirror was mounted above the camera to provide a full field view. It was attached to a circular piece of 22ga Weld Steel Sheet. Holding up this sheet were four 1/4” aluminum rods. Each rod came off of one of the four supports of the base structure. We received help bending the 1/4” aluminum rods correctly from a mechanical engineer. They were screwed to the supports in two locations each.

With only the screws at the bottom the top still had a great deal of freedom to move. So we modified our original plans. Braces made of 22ga Weld Steel Sheets were added to the poles coming up diagonally from the base supports. After these were added the top moved very little. The work on this part was then looked over by a mechanical engineer.

Figure 33 - Finished Mirror Mounting
4.7.3 Processor Mounting

The processor was planned to be mounted between the pressure tank and the cylinder. This location could work, but would be a very tight fit. The computer would be very close to the moving parts of the cylinder itself. The decision was made that this would not be best, and a new location was found.

To provide room for the processor, the pressure tank needed to be moved further towards the front of the robot. Moving it forward allowed room at the back for the processor to fit a safe distance away from the cylinder.

It was bolted on to 3 brackets made from 22ga Weld Steel Sheets. Two of the brackets attached to the frame of the robot and one attached to the base. To remove the processor these three braces must be taken off. On the back of the processor two more brackets were put into place. These two hold the processor in place but do not screw into the processor. They also have weather stripping on them to provide a cushion. There is also weather stripping below the processor to provide a cushion. The old microcontroller box rested against some of the electronics of the cylinder. A 1/4" aluminum rod was put in to prevent damage to these electronics.

![Figure 34 - Final Processor Mounting](image_url)
4.7.4 Battery Mounting

The original battery needed to be moved to make room for the new to fit below the microcontroller as well. These are both held on using webbing straps which are bolted onto the frame of the robot. In their current locations, they both fit below the microcontroller box.
4.8 Integration with Year I

The primary integration between the new design and the previous year’s design is the communication between the computer and microcontroller. This was realized by a USB to Serial cable manufactured by Sabrent, model number SBT-USC1K. The computer has multiple USB ports which can be used by the adapter, and the microcontroller has two serial ports available to use. The microcontroller already was using SCI-C for the connection to the motor drivers, so SCI-A was selected to connect to the new processor system.
5. Goals and Tested Parameters

We separated the goals and testing evaluation into two sections, the mechanical / electrical systems and the software system. The mechanical / electrical goals include requirements for RoboCup rules, power limits and safety. The software goals include being able to take pictures, distinguished colors, determine objects, determine location, and the speed that these are completed. All of these goals were presented and documented at the beginning of the semester.

5.1 Mechanical and Electrical Restrictions and Requirements

5.1.1 Weight and Size Restrictions

The vision system must fit in the size and weight restrictions given by the Middle Size Robot League Rules and Regulations. We will measure and weigh the finished robot and make any corrections that need to be made to make it fit in the guidelines.

We tested the dimensions of the robot by taking the average of three measurements (one from each member of the group) using a common household tape measure.

The regulations for dimensions are 50cm x 50cm x 80cm.

Our robot is 48cm x 49.5cm x 79.5cm, which is within our required parameters.

We tested the weight of the robot by taking the average of three measurements on the scale moved into room ECE 342.

The maximum weight allowed by RoboCup is 40 kg.

Our robot is 36.6 kg, which is under our required parameters.

5.1.2 Power Requirements

The robot must be able to maintain power and fully operate for the entire match. The match is 30 minutes, consisting of two equal periods of 15 minutes, with a possible 15 minute overtime period. We will run the robot for the length of the match. We will verify that the robot is still operation at the end of the appropriate amount of time. The robot must run 45 minutes and still be fully operational.
We ran the robot to failure five times, these times were 62, 58, 65, 55, and 70. Our robot ran an average of 62 minutes and still has full function of processing capabilities and power to USB for the camera. It is considered to have passed the power requirements.

5.1.3 Safety

The robot must be built with safety in mind. There can be nothing on the robot's vision system that could cause discomfort to the audience or damage to other robots. We will make sure that the robot does not produce waveforms that could cause people discomfort (loud noises, or intense lights).

As we are continuing on a previous year's project, we are required to understand the safety issues that may arise with the previous equipment and, if needed, provide adequate solutions to these issues.

We must also make sure that no part of the robot will come ajar, or loose during the competition.

As we do not have any signals being generated from the robot (such as wireless) at this time, there is no discomfort to or interference with outside objects. All of the added components have been secured to the robot to prevent any parts from coming loose during competition. Finally, there is a kill switch that was installed last year that has been checked and made sure that all systems are killed on the robot when the switch is thrown. This prevents the possibility of a runaway robot causing harm to outside objects.

5.2 Software Goals and Requirements

5.2.1 Identify Colors

The vision system must be able to identify orange, green, white, and black colors. We will show the robot large regions of orange, green, white, and black. We will have the robot send an output letting us know what color it sees.

We took a neutral color background (not one of the colors needed to be identified) and placed a solid color object in front of it. We then took a picture and set our code to identify the presented color. As seen in the figures below our robot was able to successfully identify all four colors: orange, green, white, and black.
5.2.2 Identify and Track Orange Soccer Ball

The vision system must be able to identify the orange soccer ball at any place on the playing field. It must be able to identify the ball on the field within 0.5 seconds when first finding the ball, or if it loses track of the ball. When following the ball, the system must be able to find the ball within 0.25 seconds 90% of the time. We will have the robot keep track of how often it knows where the ball is as we move the ball around. We will check these results to make sure that it is within the given parameters above.

We placed the ball in various positions around the robot and checked to see if the robot was able to "see" the ball. The robot was programmed to place a blue dot on where it thought the ball was. As seen on Figure 31 - Identify Orange FIFA Size 5 Soccer Ball, the robot was able to locate the ball.
The vision system must report the relative bearing and distance from the robot to the ball's current location. For error, the distance results must be within 10% or better when more than 0.5 meters away. The distance results must also be within .05 meters when the ball is less than 0.5 meters away. The angle calculated to the ball must be within 5° of the actual angle to the ball. The robot will show to the user the relative bearing and distance to the ball, and the user will check its accuracy.

To test the distance recognition we programmed the robot to output the distance in which it though the orange ball was away from the robot. As seen in the following table our robot was able to identify the distance of the ball within 10% except for the first value. This is allowed do to the increased allowance at the shorter distances. The reason for this allowance is because a change in the smaller range will inflate the error, but will have less of an effect on locating the ball.

<table>
<thead>
<tr>
<th>Actual Distance From Robot (ft)</th>
<th>Reported Distance By Robot (ft)</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.161</td>
<td>16.1%</td>
</tr>
<tr>
<td>2</td>
<td>2.002</td>
<td>0.1%</td>
</tr>
<tr>
<td>3</td>
<td>3.061</td>
<td>2.0%</td>
</tr>
<tr>
<td>4</td>
<td>3.863</td>
<td>3.4%</td>
</tr>
<tr>
<td>5</td>
<td>5.242</td>
<td>4.8%</td>
</tr>
</tbody>
</table>
To test the direction from the robot we followed a similar procedure as the distance calculations. We programmed to robot to display the angle at which it had calculated the balls position as. In the table below we show that the robot was within 3.9° on average of the balls known value. This gives approximately 1.08% error in known direction which is well within our parameters. The figure below also shows that the robot is able to locate the ball in any direction around the robot, accomplishing our goal of having an omnidirectional vision system.

Table 10 - Direction reported by robot

<table>
<thead>
<tr>
<th>Actual Angle from Kicking Arm (°)</th>
<th>Reported Angle By Robot (°)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.026</td>
</tr>
<tr>
<td>45</td>
<td>34.131</td>
</tr>
<tr>
<td>90</td>
<td>82.248</td>
</tr>
<tr>
<td>135</td>
<td>127.145</td>
</tr>
<tr>
<td>180</td>
<td>174.460</td>
</tr>
<tr>
<td>225</td>
<td>225.361</td>
</tr>
<tr>
<td>270</td>
<td>273.112</td>
</tr>
<tr>
<td>315</td>
<td>318.547</td>
</tr>
</tbody>
</table>
5.2.4 Maximum Vision Distance

The vision system must be able to see the entire field at once. We will put the robot at a stationary location, an object 22m away and check if the object is visible in the image the camera produces.

The maximum distance the robot was able to accurately locate the soccer ball was 6m. This is well short of our goal of 22m, but after review of the requirement, we found that 22m was too far of a distance.

There will be a total of five robots on a team so one robot does not need to individually see the whole field. If there was a maximum distance of 22m, the robot would not be utilizing the entire vision and would waste processing power. With the distance of 6m, if the robot is located centrally on the field, it will be able to see over 50% of the field. This is shown in the figure below.

Figure 42- Vision of soccer ball in 360°
5.2.5 Integration with Previous Software/Hardware and Use of Kicking Sensor

There is a sensor designed to locate the ball when it is near the kicking arm. The system for locating the ball should be able to use this data, as well as the data for locating the ball. We will put the ball next to and around the kicking arm, keeping track of how often the robot realizes the ball is within kicking distance. We will check these results to make sure that it has a 95% accuracy.

We were not able to accomplish this goal. There were communication errors between the processing unit and the microprocessor. We were able to send data via RS-232 serial to the microprocessor, and the microprocessor was able to send and receive data, but the data was incorrect. We tested this buy first creating a loopback feed with the RS-232 connection on the processor side. We were able successfully send a byte of data and receive that same byte of data. We did the same thing with the microprocessor using the code from last year’s group. Next we tried to send two bytes of data from the processor to the microprocessor, have the microprocessor add the two bytes together and send the resulting byte back. We were not able to receive the correct byte and ran out of time to finish this step.
We have a few recommendations for next year. First, start the design semester by fully reviewing the current robots systems. Make sure that the power works, there are no shorts in the electrical wiring, the robot runs as it should, and record all this information as it is taken. This will allow the design phase to account for any modifications or corrections needed to continue on. We did not do this until well into the second semester. We found that several of the robots electrical systems where non-functional and needed to be repaired to fully complete our project. This caused us to underestimate the amount of work needed and required extra resources to complete the goals.

Have a significant part of your design be focused on bringing all three years of progress together at the end of your year. Most of the work done by the first two years focused only individual parts of the project. At some point it all needs to work together and that will take time and planning.

Leave room in the budget for “extras”. This is where our budget when the most over. We had planned to spend $1500, but did not allow enough for mounting supplies and rebuilding supplies. Our “extras“ cost of $200 put us over budget for the project.

Familiarize yourself with the software of the previous years. As it was written by somebody else, it takes time to read, and fully understand how the code works, and what needs to be modified to work for your goals.

Finally, make sure that the parts are ordered at the end of the first semester. This will allow you to start building at the beginning of the second semester, saving valuable weeks in the build phase. We were pushed behind schedule as we were waiting for parts 2 months into the second semester.
In conclusion the robot is now equipped with an omnidirectional vision system. The major requirements of the vision system have been met. It is able to constantly see in a 360° view. The robot is able to distinguish the four main colors needed to compete at RoboCup: orange, white, green, and black. As well as being able to see colors, the robot can also determine which object is the orange soccer ball. The robot can report the ball’s location via direction and distance through a serial port as well. All of this is accomplished autonomously; the robot has self-sufficient power and software capabilities to achieve its requirements without the external directions.

In addition it has an interface that allows users to see what the robot is seeing. It also shows what the robot is “thinking”. It can display where it thinks the ball is on the image and where it thinks it is going. It can also display where it thinks different colors are located in its field of view.
8. References

Intel DQ45EK Product Guide

Middle Size Robot League Rules and Regulations for 2009

Mightex Systems
http://www.mightexsystems.com/

Mini-ITX.com, M2-ATX-Manual

Newegg.com
http://www.newegg.com

RoboRealm Tutorials, Edge Detection Picture
http://www.roborealm.com/tutorial/Obstacle_Avoidance/slide020.php

RoboRealm Tutorials, RGB Filtering

TigerDirect.com
http://www.tigerdirect.com

Wikipedia, Hard Disk Drive
#include <stdio.h>

#define COM1 0;
#define COM2 1;
#define COM3 2;

char COM_PORT[] = "COM2";

unsigned char byte1;
unsigned char byte2;
unsigned char recByte;
bool reedExtended = false;
bool reedRetracted = false;
bool proximity = false;
bool programSwitch = false;

void sendByte(unsigned char value);
void kickNow(bool value);
void kickFree(bool value);
void move(unsigned char speed, int degrees);
void turn(unsigned char value);
void updateMicro(void);
void stop(void);
void serialSend(unsigned char value);
unsigned char serialRec(void);
void serialInit(void);

HANDLE COM_IO;
BYTE bResult;
BYTE bRead=-1;
BYTE bData;
DWORD bWritten=0;
DCB dcb;

void kickNow(bool value){ //Tell microcontroller to kick, independent of the prox sensor
  if(value){
    byte2 = byte2 | 0x01;
  } else {
    byte2 = byte2 & 0xFE;
  }
  updateMicro();
  if(value){
    byte2 = byte2 | 0x01;
  } else {
    byte2 = byte2 & 0xFE;
  }
  updateMicro();
}
void kickFree(bool value){        //Update microcontroller to allow kicking
    if(value){
        byte2 = byte2 | 0x02;
    } else {
        byte2 = byte2 & 0xFD;
    }
    updateMicro();
}

void move(unsigned char speed, int degrees){     //Update bytes to go at
    unsigned char temp1 = (degrees%360)/2;
    unsigned char temp2 = speed>>4;

    temp1 = (degrees%360)/2;
    temp1 = (temp1<<2) & 0xFC;
    byte2 = (byte2 & 0x03) | temp1;

    temp2 = speed & 0xF0;
    byte2 = (byte2 & 0x0F) | temp2;
}

void turn(int value){                      // 0 <= value <= 15
    unsigned char temp = (((value%16) & 0x0F) << 2) & 0x3C;
    byte2 = (byte2 & 0xC3) | temp;
}

void turn(int value,int direction){       // 0 <= value <= 15
    unsigned char temp = (((value%16) & 0x0F) << 2) & 0x3C;
    byte2 = (byte2 & 0xC3) | temp;
}

void stop(void){    //Update bytes to send to microcontroller to stop
    byte1 = 0;
    byte2 = 0;
}

void updateMicro(void){    //Send commands to microcontroller
    serialSend(byte1);
    Sleep(100);
    serialSend(byte2);
    Sleep(100);
    recByte = serialRec();

    //Get switch statuses from received byte from microcontroller
    programSwitch = (recByte & 0x80) == 0x80;
    reedExtended = (recByte & 0x04) == 0x04;
    reedRetracted = (recByte & 0x02) == 0x02;
    proximity = (recByte & 0x01) == 0x01;
void serialSend(unsigned char value) {  //Write one byte to serial port
    WriteFile(COM_IO, &value, 1, &bWritten, NULL);
}

unsigned char serialRec(void) {  //Read one byte from serial port
    unsigned char recByte = -1;
    DWORD nByteRead = 0;
    while (nByteRead == 0) { //wait until there is a valid byte to be read
        ReadFile(COM_IO, &recByte, 1, &nByteRead, NULL);
    }
    return recByte;
}

void sendByte(unsigned char value) {
    serialSend(value);
}

void serialInit(void) {
    COM_IO = CreateFile("COM2", GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING, 0, 0);
    dcb.BaudRate = CBR_19200; //Set baud rate to 19200 bps
    dcb.ByteSize = 8;  //Set byte size to 8 bits
    SetCommState(COM_IO, &dcb);
}

#include "def.h"
#include "Functions.c"
#include "glutfunctions.cpp"
#include "serialcontrol.cpp"

#ifdef SKIP_MICRO
#include "movement.c"
#endif

#ifndef __DEF_H__
#define CONTRAST 50    //0-100
#define BRIGHT 50     //0-100
#define SHARPNESS 3    //0: No Sharp, 1: Sharp, 2: Sharper, 3: Sharpest
#define RedGain 8
#define GreenGain 8
#define BlueGain 8
#define MY_BUFSIZE 1024 // Buffer size for console window titles.
#endif

void exit(void);
void pictureCallback(TProcessedDataProperty* attributes, unsigned char *Data);
HWND GetConsoleHwnd(void);
#endif
void main(int argc, char** argv){
    DWORD nByteRead=0;
    byte1 = 0x01;
    byte2 = 0x02;
    bData = byte1;
    recByte = 0x03;
    #ifdef SERIAL
    serialInit();
    Sleep(2);
    #ifdef SKIP_MICRO
    serialSend(0xAA);
    Sleep(4);
    #endif
    #endif

    #ifdef SERIAL_LOOP
    serialSend(128);
    serialSend(1);
    serialSend(128);
    serialSend(((128+1+128) & 127));
    while(1){
        updateMicro();
        //serialRec();
        serialSend(0x13);
    }
    //while(1){updateMicro();byte1++;byte2++;Sleep(1000);}
    while(1){
        printf("Enter first hex value (00-FF): ");
        scanf("%x", &byte1);
        printf("Enter second hex value (00-FF): ");
        scanf("%x", &byte2);
        printf("Entered values: %d+%d should be %d\n",byte1,byte2,byte1+byte2);
        updateMicro();
        //recByte = serialRec();
        //printf("Received value is: %x\n\n",recByte);
    }
    while(1){
        serialSend(bData);
        Sleep(200);
        recByte = serialRec();
        printf("Received byte: %x\n\n",recByte);
        //CLOSING HANDLE!!!!!!!
    }
}
//CloseHandle(COM1_IO);

bData++;

#endif

#ifdef GLUT_ACTIVE
    //Open files
    init_window(argc,argv);
    //Initialize GLUT window

    main2();
    other_init();
#endif

pictures taken for the filename
    glutDisplayFunc(display); //Set refresh event
    glutKeyboardFunc(keyboard); //Set keyboard event

    glutTimerFunc(10,timedImage,1); //Get a new image after a period of time

    glLoadIdentity(); //Load identity matrix
    glutMainLoop(); //Start event-driven programming

#endif

void main2(void){
    int c=0;

    c=BUFUSB_InitDevice();
    c=BUFUSB_GetModuleNoSerialNo(1,ModuleNo,SerialNo);
    //c=BUFUSB_InstallFrameHooker(1,pictureCallBack);
    c=BUFUSB_InstallFrameHooker(1,NULL); //Run in BMP mode
    c=BUFUSB_AddDeviceToWorkingSet(DEVICEID); //Add camera 1 to list of active cameras

    d=GetConsoleHwnd();
    c=BUFUSB_StartCameraEngine(d,8); //Start cameras
    c=BUFUSB_SetCameraWorkMode(DEVICEID,0); //Set camera 1 to work in continuous mode.
    c=BUFUSB_StartFrameGrab(0x8888); //Get frames continuously
    c=BUFUSB_SetResolution(DEVICEID,CAMERA_RES,0,1); //Set camera 1 to 1280x1024, no decimation, 1 frames in buffer
    c=BUFUSB_SetExposureTime(DEVICEID,1200); //Set camera 1 to 20000us (2nd x 50us)
    c=BUFUSB_SetXYStart(DEVICEID,X_OFFSET,Y_OFFSET);
    //Set offset of camera 1 to x=0, y=0
    c=BUFUSB_SetGamma(GAMMA*10,CONTRAST,BRIGHT,SHARPNESS);
    c=BUFUSB_InstallUSBDeviceHooker(CameraFaultCallBack);

#ifdef PRINT_MENU
    printMenu();
#endif
#endif

readImage();
}

void readImage(void){
    BUFUSB_GetCurrentFrame(1,DEVICEID,location);
    Attributes = (TProcessedDataProperty*) location;
    MakeBlobs();
    BallLocation();
}

#ifdef SERIAL
    if(imageControlMovement)
      move(((distance*12.0/39.0/22.0)>1?0xFF:0xFF*(distance*12.0/39.0/22.0)),
           angle*180/PI);
      updateMicro();
    }
#endif

if(drawMode=='r'){    //Real image
    for(int n=0;n<ROW;n++){
        for(int m=0;m<COLUMN;m++){
            if(orange[n][m]==1){
                bitmapImage[n][m][0]=255;
                bitmapImage[n][m][1]=100;
                bitmapImage[n][m][2]=30;
            } else {
                bitmapImage[n][m][0]=128;
                bitmapImage[n][m][1]=128;
                bitmapImage[n][m][2]=128;
            }
        }
    }
} else if(drawMode=='o'){  //Orange blob image
    for(int n=0;n<ROW;n++){
        for(int m=0;m<COLUMN;m++){
            if(orange[n][m]==1){
                bitmapImage[n][m][0]=255;
                bitmapImage[n][m][1]=100;
                bitmapImage[n][m][2]=30;
            } else {
                bitmapImage[n][m][0]=128;
                bitmapImage[n][m][1]=128;
                bitmapImage[n][m][2]=128;
            }
        }
    }
} else if(drawMode=='g'){  //Green blob image
    for(int n=0;n<ROW;n++){
        for(int m=0;m<COLUMN;m++){
            if(green[n][m]==1){
                bitmapImage[n][m][0]=0;
                bitmapImage[n][m][1]=255;
                bitmapImage[n][m][2]=0;
            } else {
                bitmapImage[n][m][0]=128;
                bitmapImage[n][m][1]=128;
                bitmapImage[n][m][2]=128;
            }
        }
    }
}
} 
} 
} 
} 
else if(drawMode=='b'){  //Black blob image
  for(int n=0;n<ROW;n++)
    for(int m=0;m<COLUMN;m++)
      if(black[n][m]==1){
        bitmapImage[n][m][0]=0;
        bitmapImage[n][m][1]=0;
        bitmapImage[n][m][2]=0;
      } else {
        bitmapImage[n][m][0]=128;
        bitmapImage[n][m][1]=128;
        bitmapImage[n][m][2]=128;
      }
}

} 
} 
} 
} 
else if(drawMode=='w'){  //White blob image
  for(int n=0;n<ROW;n++)
    for(int m=0;m<COLUMN;m++)
      if(white[n][m]==1){
        bitmapImage[n][m][0]=255;
        bitmapImage[n][m][1]=255;
        bitmapImage[n][m][2]=255;
      } else {
        bitmapImage[n][m][0]=128;
        bitmapImage[n][m][1]=128;
        bitmapImage[n][m][2]=128;
      }
}

} 
} 
} 
} 
else if(drawMode=='c'){  //Combination of all colors
  for(int n=0;n<ROW;n++)
    for(int m=0;m<COLUMN;m++)
      if(orange[n][m]==1){
        bitmapImage[n][m][0]=255;
        bitmapImage[n][m][1]=100;
        bitmapImage[n][m][2]=30;
      } else if(black[n][m]==1){
        bitmapImage[n][m][0]=0;
        bitmapImage[n][m][1]=0;
        bitmapImage[n][m][2]=0;
      } else if(white[n][m]==1){
        bitmapImage[n][m][0]=255;
        bitmapImage[n][m][1]=255;
        bitmapImage[n][m][2]=255;
      } else if(green[n][m]==1){
        bitmapImage[n][m][0]=0;
        bitmapImage[n][m][1]=255;
        bitmapImage[n][m][2]=0;
      } else {
        bitmapImage[n][m][0]=128;
        bitmapImage[n][m][1]=128;
        bitmapImage[n][m][2]=128;
      }
}
}  //Translate bitmap image to the image to be displayed on the screen
for(int n=0;n<DROW;n++){
    for(int m=0;m<DCOLUMN;m++){
        ifndef FLIP
            displayImage[n][DCOLUMN-1-m][0]=bitmapImage[n*DLEVEL][m*DLEVEL][0];
            displayImage[n][DCOLUMN-1-m][1]=bitmapImage[n*DLEVEL][m*DLEVEL][1];
            displayImage[n][DCOLUMN-1-m][2]=bitmapImage[n*DLEVEL][m*DLEVEL][2];
        #else
            displayImage[m][n][0]=bitmapImage[n*DLEVEL][m*DLEVEL][0];
            displayImage[m][n][1]=bitmapImage[n*DLEVEL][m*DLEVEL][1];
            displayImage[m][n][2]=bitmapImage[n*DLEVEL][m*DLEVEL][2];
        #endif
    }
}

#ifdef PREDICT
    StoreOldLocations();
    LocationPrediction();
#endif

#ifndef FLIP
    for(int i=0;i<9;i++){
        for(int k=0;k<9;k++){  //Display current ball position on displayed image
            displayImage[(orangeRowLocation/DLEVEL)+i][(orangeColLocation/DLEVEL)+k][0]=0;
            displayImage[(orangeRowLocation/DLEVEL)+i][(orangeColLocation/DLEVEL)+k][1]=80;
            displayImage[(orangeRowLocation/DLEVEL)+i][(orangeColLocation/DLEVEL)+k][2]=255;
        }
    }
#else
    #ifdef PREDICT_DISPLAY
        //Display future ball position on displayed image
        if(orangeRowFutureLocation>=0&&orangeRowFutureLocation<ROW &&
            orangeColFutureLocation>=0&&orangeColFutureLocation<COLUMN){
            displayImage[(orangeColFutureLocation/DLEVEL)+k][(orangeRowFutureLocation/DLEVEL)+i][0]=200;
            displayImage[(orangeColFutureLocation/DLEVEL)+k][(orangeRowFutureLocation/DLEVEL)+i][1]=30;
            displayImage[(orangeColFutureLocation/DLEVEL)+k][(orangeRowFutureLocation/DLEVEL)+i][2]=255;
        }
    #endif
#endif

else
for(int i=-1;i<9;i++){
    for(int k=-1;k<9;k++){
        //Display current ball position on displayed image
        displayImage[(orangeColLocation/DLEVEL)+k][(orangeRowLocation/DLEVEL)+i][0]=0;
        displayImage[(orangeColLocation/DLEVEL)+k][(orangeRowLocation/DLEVEL)+i][1]=80;
        displayImage[(orangeColLocation/DLEVEL)+k][(orangeRowLocation/DLEVEL)+i][2]=255;
    }
}
#endif PREDICT_DISPLAY
    //Display future ball position on displayed image
    if(orangeRowFutureLocation>=0&&orangeRowFutureLocation<ROW
            &&orangeColFutureLocation>=0&&orangeColFutureLocation<COLUMN){
        displayImage[(orangeColFutureLocation/DLEVEL)+k][(orangeRowFutureLocation/DLEVEL)+i][0]=200;
        displayImage[(orangeColFutureLocation/DLEVEL)+k][(orangeRowFutureLocation/DLEVEL)+i][1]=30;
        displayImage[(orangeColFutureLocation/DLEVEL)+k][(orangeRowFutureLocation/DLEVEL)+i][2]=255;
    }
#endif

void exit2(void){
    BUFUSB_StopCameraEngine();
    BUFUSB_UnInitDevice();
    exit(0);
}

void pictureCallBack(TProcessedDataProperty* attributes,unsigned char *Data){
    //printf("Picture read.\n");
    readImage();
    i++;
void CameraFaultCallBack(int ImageType)
{
    #ifdef PRINT
        printf("Camera Failure\n\n");
    #endif
    exit2();
    Sleep(3000);
}

#define color(int color, int r, int c)
    return *(location+56+((color*1280*1024)+(1280*r)+(c)));

void BMPHeader(FILE* outfile)
{
    unsigned int filesize=125829174;//125,829,174 bytes 7800036
    unsigned int offset=54;
    #ifdef SMALL
        char buffer[] = {'B','M',  //Magic word
                        54,00,120,0,    //File Size  125,829,174
                        00,00,      //Reserved - 0
                        00,00,      //Reserved - 0
                        54,00,00,00,    //Offset to image data
                        00110110 -> 11011000
                        40,00,00,00,    //40 00101000 -> 10100000
                        00,05,00,00,    //Width  1280 ->
                        0x500
                        00,04,00,00,    //Height 1024 -> 0x400
01,00,    //Image planes
24,00,    //Bits per pixel
00,00,00,00,    //Compression
00,00,120,0,    //Image data size
20,00,00,00,    //pixels per meter horizontal
20,00,00,00,    //pixels per meter vertical
00,00,00,00,    //colors in image
00,00,00,00};    //important colors in image
#else
    char buffer[] = {'B','M',    //Magic word
54,0,0,18,//54,00,128,07,    //File Size
00,0,    //Reserved - 0
00,0,    //Reserved - 0
54,00,00,00,    //Offset to image data
40,00,00,00,    //40
0,(int)8,0,0,//00,08,00,00,    //Height
2048 -> 0x800
0,06,0,0,//00,06,00,00,    //image data size
20,00,00,00,    //pixels per meter horizontal
20,00,00,00,    //pixels per meter vertical
00,00,00,00,    //colors in image
00,00,00,00};    //important colors in image
#endif

void printBMP(char *filename){
    FILE* output = fopen(filename,"wb");
    BMPHeader(output);
    //for(int k=ROW_END-1;k>=ROW_START;k--){
    //    for(int n=COL_START;n<COL_END;n++){
    //        fprintf(output,"%2x%2x%2x",color(2,k,n),color(1,k,n),color(0,k,n));
    //        fprintf(output,"%2x%2x%2x",bitmapImage[k][n][0],bitmapImage[k][n][1],bit
    //mapImage[k][n][2]);
    //    }
    //}
}
#ifdef SMALL
for(int i=0; i<ROW; i++){
    for(int k=0; k<COLUMN; k++){
        bitmapImage2[i][COLUMN-1-k][0]=bitmapImage[i][k][2];
        bitmapImage2[i][COLUMN-1-k][1]=bitmapImage[i][k][1];
        bitmapImage2[i][COLUMN-1-k][2]=bitmapImage[i][k][0];
    }
}  
fwrite(bitmapImage2,2,ROW*COLUMN*3,output);
#else
for(int i=0; i<ROW; i++){
    for(int k=0; k<COLUMN; k++){
        bitmapImage2[k][i][0]=bitmapImage[i][k][2];
        bitmapImage2[k][i][1]=bitmapImage[i][k][1];
        bitmapImage2[k][i][2]=bitmapImage[i][k][0];
    }
}  
fwrite(bitmapImage2,6,ROW*COLUMN,output);
#endif
fclose(output);

void createArray(void){
    for(int p=0; p<(ROW); p++){
        for(int q=0; q<(COLUMN); q++){
            for(int r=0; r<3; r++){
                imageData[p][q][r]=*(location+56+(p*COLUMN*3+q*3+r));
            }
        }
    }
}

void printArray(char *filename){
    FILE* output = fopen(filename,"w");
    for(int k=ROW_START; k<ROW_END/8; k++){
        for(int n=COL_START; n<COL_END/8; n++){
            fprintf(output,"%2x%2x%2x
",bitmapImage[k][n][0],bitmapImage[k][n][1],bitmapImage[k][n][2]);
        }
    }
    fprintf(output,"\n");
    fclose(output);
}

void printMenu(void){
    printf("Key Menu:\n");
    printf("  r - Real Image\n");
    printf("  o - Orange Blobs\n");
    printf("  w - White Blobs\n");
    printf("  b - Black Blobs\n");
printf("  g - Green Blobs\n");
printf("  c - All Blobs\n");
printf("  p - Pause Image\n");
printf("  s - Save Image to File\n");
printf("  q - Quit\n");
}

9.3 def.h

#ifndef __DEF_H__
#define __DEF_H__

#define FLIP
#define SMALL       //Use 1280x1024 resolution image
//#define PRINT
//#define PRINT_DIST
#define PREDICT     //Calculates predicted future ball location
//#define PREDICT_DISPLAY   //Displays predicted ball location on image
#define PRINT_MENU
#define GLUT_ACTIVE
#define SERIAL
#define PRINT_SERIAL
//#define SERIAL_LOOP
//#define SKIP_MICRO

#include<stdio.h>
#include<cstdio>
#include<stdlib.h>
#include<windows.h>

#ifndef PREDICT_DISPLAY
#define PREDICT
#endif
#define PRINT
#endif
#define SMALL
#ifndef FLIP
#define FLIP
#endif
else
# undef FLIP
#endif
#endif

#include<string.h>
#include<math.h>
#include<GL\glut.h>
#include<BUF_USBCamera_SDK.h>

#ifndef SMALL
#define ROW 1024
#define COLUMN 1280
#define DLEVEL 2
#define CAMERA_RES 8
#define GLUT_TIMER 1
#else
#define ROW 2048
#define COLUMN 1536
#define DLEVEL 4
#define CAMERA_RES 10
#define GLUT_TIMER 8
#endif

#define DROW ROW/DLEVEL
#define DCOLUMN COLUMN/DLEVEL
#define DEVICEID 1
#define GAMMA 1
#define CONTRAST 50      //0-100
#define BRIGHT 65         //0-100
#define SHARPNESS 3       //0: No Sharp, 1: Sharp, 2:
                       //3: Sharpest
#define RedGain 54        //1-64
#define GreenGain 59      //1-64
#define BlueGain 64       //1-64
#define MY_BUFSIZE 1024   // Buffer size for console window titles.
#define X_OFFSET 384      //(2592+ROW)/2
#define Y_OFFSET 240      //(1944+COLUMN)/2

//--Print Range-------------------------------------------------
#define ROW_START 0      //0
#define ROW_END ROW-ROW_START-1 //1023
#define COL_START 0      //0
#define COL_END COLUMN-COL_START-1 //1279
#define PI 3.1415926535

//CameraTest function prototypes
void exit2(void);
void pictureCallBack(TProcessedDataProperty* attributes,unsigned char *Data);
HWND GetConsoleHwnd(void);
void CameraFaultCallBack(int ImageType);
byte color(int color, int r, int c);
void BMPHeader(FILE* outfile);
void printBMP(char *filename);
void createArray(void);
void printArray(char *filename);
void main2(void);
void readImage(void);
void printMenu(void);
void StoreOldLocations(void);
void LocationPrediction(void);

//CameraTest global variables
int i=0;
TProcessedDataProperty* Attributes;
HWND d;
char ModuleNo[16];
char SerialNo[16];
GLubyte imageData[ROW][COLUMN][3];
byte* location = (byte *)imageData;

//Functions function prototypes
void MakeBlobs(void);
void BallLocation(void);

//GlutFunctions function prototypes
void timedImage(void);
void other_init(void);
void display(void);

//Functions variables
int white[ROW][COLUMN];
int black[ROW][COLUMN];
int green[ROW][COLUMN];
int orange[ROW][COLUMN];

int headerOffset = 56;

//---------Row and Column Sum arrays
int orangeTotal = 0;

//----variables for calculating future locations
int framesPerSecond = 20;
const int thingsStored = 50;
double locationList[2][thingsStored];
int storageCounter = -1;

int orangeRowLocation = 0;
int orangeColLocation = 0;
int orangeRowFutureLocation = 0;
int orangeColFutureLocation = 0;

//--------------------------------
//GlutFunctions Variables
char pictureName[] = "output00.bmp";  //Output picture name
bool imageControlMovement = false;  //Flag to move robot based on image read

#endif
extern int headerOffset;
extern unsigned char* location;
extern char pictureName[];
int picCnt = 0;
int tempSpeed = 0xFF;

extern void stop(void);
extern void updateMicro(void);
extern void move(unsigned char speed, int degrees);
extern void Rotate(int rot, int speed);
extern void kickFree(bool);

extern unsigned char byte1;
extern unsigned char byte2;

GLubyte bitmapImage[ROW][COLUMN][3];
#ifdef SMALL
GLubyte bitmapImage2[ROW][COLUMN][3];
#else
GLubyte bitmapImage2[COLUMN][ROW][3];
#endif
#ifdef FLIP
GLubyte displayImage[ROW][COLUMN][3];
#else
GLubyte displayImage[COLUMN][ROW][3];
#endif
char drawMode = 'c';
bool draw = true;
bool dec = false;
int rowNum;
int frameCount;
int lastX=0;
int lastY=0;

// Other GLUT initializations
void other_init() {
    #ifdef GLUT_ACTIVE
        rowNum=0;
        frameCount=0;
        glClearColor(0.0, 0.0, 0.0, 1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
    #ifndef FLIP
        glOrtho(0,COLUMN,0,ROW,-2,1);
    #else
        glOrtho(0,ROW,0,COLUMN,-2,1);
    #endif
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}
```c
#include <GL/glut.h>

// Draw buffer

void timedImage(int v) {
    glClear(GL_COLOR_BUFFER_BIT);
    readImage();    // Read new image and display on screen
    if (frameCount++ == 1) {
        frameCount = 0;
        if (draw) {
            glDrawPixels(COLUMN, ROW, GL_RGB, GL_UNSIGNED_BYTE, displayImage);
        }
    } else {
        gluttimerFunc(GLUT_TIMER, timedImage, v+1); // Set a new timer to get a new image
    }
}

// Initialize GLUT Window
void init_window(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(DCOLUMN, DROW);
    glutInitWindowPosition(200, 0);
    glutCreateWindow("Camera Image");
}

// Refresh callback function
void display(void) {
    if (GLUT_ACTIVE)
        glClear(GL_COLOR_BUFFER_BIT);
    readImage();
    if (frameCount++ == 1) {
        frameCount = 0;
        if (draw) {
            glDrawPixels(COLUMN, ROW, GL_RGB, GL_UNSIGNED_BYTE, displayImage);
        } else {
            glDrawPixels(ROW, COLUMN, GL_RGB, GL_UNSIGNED_BYTE, displayImage);
        }
    } else {
        glutSwapBuffers();
    }
}

void keyboard(unsigned char key, int x, int y) {
    if (key=='P'|key=='p') { // Pause display
        draw=!draw;
        glutPostRedisplay();
    }
}
```

---

```c
// Draw buffer

void timedImage(int v) {
    glClear(GL_COLOR_BUFFER_BIT);
    readImage();    // Read new image and display on screen
    if (frameCount++ == 1) {
        frameCount = 0;
        if (draw) {
            glDrawPixels(COLUMN, ROW, GL_RGB, GL_UNSIGNED_BYTE, displayImage);
        }
    } else {
        gluttimerFunc(GLUT_TIMER, timedImage, v+1); // Set a new timer to get a new image
    }
}

// Initialize GLUT Window
void init_window(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(DCOLUMN, DROW);
    glutInitWindowPosition(200, 0);
    glutCreateWindow("Camera Image");
}

// Refresh callback function
void display(void) {
    if (GLUT_ACTIVE)
        glClear(GL_COLOR_BUFFER_BIT);
    readImage();
    if (frameCount++ == 1) {
        frameCount = 0;
        if (draw) {
            glDrawPixels(COLUMN, ROW, GL_RGB, GL_UNSIGNED_BYTE, displayImage);
        } else {
            glDrawPixels(ROW, COLUMN, GL_RGB, GL_UNSIGNED_BYTE, displayImage);
        }
    } else {
        glutSwapBuffers();
    }
}

void keyboard(unsigned char key, int x, int y) {
    if (key=='P'|key=='p') { // Pause display
        draw=!draw;
        glutPostRedisplay();
    }
}```
} else if(key=='O'||key=='o'){
    //Orange
    drawMode='o';
    glutPostRedisplay();
} else if(key=='G'||key=='g'){
    //Green
    drawMode='g';
    glutPostRedisplay();
} else if(key=='B'||key=='b'){
    //Black
    drawMode='b';
    glutPostRedisplay();
} else if(key=='W'||key=='w'){
    //White
    drawMode='w';
    glutPostRedisplay();
} else if(key=='C'||key=='c'){
    //All blob processed images
    drawMode='c';
    glutPostRedisplay();
} else if(key=='R'||key=='r'){
    //Raw image
    drawMode='r';
    glutPostRedisplay();
} else if(key=='+'||key=='='){
    //Increase speed
    if(tempSpeed<(0xFF-5)){
        tempSpeed+=5;
    }
    else if(key=='-'){
        //Decrease speed
        if(tempSpeed>5){
            tempSpeed-=5;
        }
    } else if(key=='I'||key=='i'){
        //Toggle movement control from image
        imageControlMovement = !imageControlMovement;
    } else if(key=='S'||key=='s'){
        //Save current image
        pictureName[0]=0;
        picCnt=picCnt+1;
        pictureName[6]='0'+(picCnt/10)%100;
        pictureName[7]='0'+picCnt%10;
        printBMP(pictureName);
        #ifdef SMALL
            picCnt=picCnt+1;
            pictureName[6]='0'+(picCnt/10)%100;
            pictureName[7]='0'+picCnt%10;
            printBMP(pictureName);
        #endif
        glutPostRedisplay();
    } else if(key=='Q'||key=='q'){
        //Quit
        exit2();
    } else if(key=='5'){
        //Stop
        stop();
        move(0,0);
        updateMicro();
    } else if(key=='8'){
        //Move
        move(tempSpeed,0);
        updateMicro();
    } else if(key=='2'){
        //Move
        backward
    }
else if (key=='4') {
    move(tempSpeed, 135);
    updateMicro();
} else if (key=='6') {
    move(tempSpeed, 45);
    updateMicro();
}

else if (key=='k' || key=='K') {
    kickFree(!(byte2&0x02));
}

#endif
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Diagram of Connection between Components</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Motherboard Layout</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Intel E8500 Processor</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Standard Hard Drive</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>SATA Solid-State Hard Drive</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>Apex MW-100 Mini-ITX Case</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>Mounting Position of the Case</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>Voltage Regulator Layout</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>BCE-C030-U Enclosed Camera</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>BCE-C030-U Technical Drawing</td>
<td>15</td>
</tr>
<tr>
<td>11</td>
<td>H2Z0414C-MP Lens Technical Drawing</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>Scientific 7-1301-K Spherical Convex Mirror</td>
<td>17</td>
</tr>
<tr>
<td>13</td>
<td>Tangent line to where the ray hits the curved mirror</td>
<td>18</td>
</tr>
<tr>
<td>14</td>
<td>Reflection Angle</td>
<td>18</td>
</tr>
<tr>
<td>15</td>
<td>Mirror Calculation Diagram</td>
<td>19</td>
</tr>
<tr>
<td>16</td>
<td>Triangle to calculate w</td>
<td>20</td>
</tr>
<tr>
<td>17</td>
<td>Triangle to calculate r</td>
<td>21</td>
</tr>
<tr>
<td>18</td>
<td>Image to calculate distance between camera and mirror</td>
<td>22</td>
</tr>
<tr>
<td>19</td>
<td>Mirror backing plate design</td>
<td>24</td>
</tr>
<tr>
<td>20</td>
<td>Mirror rod design</td>
<td>25</td>
</tr>
<tr>
<td>21</td>
<td>RGB Filtering</td>
<td>25</td>
</tr>
<tr>
<td>22</td>
<td>Edge Detection</td>
<td>26</td>
</tr>
<tr>
<td>23</td>
<td>Determining future position of an object</td>
<td>27</td>
</tr>
<tr>
<td>24</td>
<td>Processor and Camera</td>
<td>28</td>
</tr>
<tr>
<td>25</td>
<td>Apex MW-100 Mini-ITX Case</td>
<td>29</td>
</tr>
<tr>
<td>26</td>
<td>Transcend 2.5&quot; SATA-2 SSD TS6GSSD25S-S Solid State Hard Drive</td>
<td>29</td>
</tr>
<tr>
<td>27</td>
<td>Dynatron Low Profile CPU Heat sink/Fan</td>
<td>29</td>
</tr>
<tr>
<td>28</td>
<td>Amstron 35Ah 12V Battery</td>
<td>31</td>
</tr>
<tr>
<td>29</td>
<td>Werker 10Ah 12V Battery</td>
<td>32</td>
</tr>
<tr>
<td>30</td>
<td>Mightex BCE-C030-U Color CMOS Camera</td>
<td>33</td>
</tr>
<tr>
<td>31</td>
<td>Robot with reflective half of a spherical chrome top light bulb temporarily attached</td>
<td>34</td>
</tr>
<tr>
<td>32</td>
<td>Camera mounted</td>
<td>38</td>
</tr>
<tr>
<td>33</td>
<td>Finished Mirror Mounting</td>
<td>39</td>
</tr>
<tr>
<td>34</td>
<td>Final Processor Mounting</td>
<td>40</td>
</tr>
<tr>
<td>35</td>
<td>Final Processor Mounting</td>
<td>41</td>
</tr>
<tr>
<td>36</td>
<td>Final Battery Mounting</td>
<td>42</td>
</tr>
<tr>
<td>37</td>
<td>Identify Orange</td>
<td>45</td>
</tr>
<tr>
<td>38</td>
<td>Identify Green</td>
<td>45</td>
</tr>
<tr>
<td>39</td>
<td>Identify White</td>
<td>46</td>
</tr>
<tr>
<td>40</td>
<td>Identify Black</td>
<td>46</td>
</tr>
</tbody>
</table>
11. List of Tables

Table 1 - Motherboard Connections................................................................. 7  
Table 2 - Main Power Connector Pinout.......................................................... 12  
Table 3 - Processor Core Power Connector Pinout.......................................... 12  
Table 4 - Voltage Regulator Connections....................................................... 13  
Table 5 - Calculations of w and focal length with h at 3 cm ......................... 23  
Table 6 - Calculations of w and focal length with h at 5 cm ......................... 23  
Table 7 - Calculations of w and focal length with h at 8 cm ......................... 23  
Table 8 - Calculated values for Ginsberg Scientific 7-1301-K Spherical Convex Mirror ................................................................. 24  
Table 9 - % Error in Distance Calculations .................................................... 47  
Table 10 - Direction reported by robot........................................................... 48