Ball/Hole Locator Module: A Junior Design Project.

Team A

Kurt Caviggia

Jonathan Deming

Azmat Bhatty

EE 382 Introduction to Design

Spring 2002

Dr. Wedeward

Dr. Rison

Abstract:

The New Mexico Tech Electrical Engineering Department Introduction To Design course set forth the design problem of building a fully autonomous golfing robot. The robot is designed to receive GPS coordinates of the Ball and Hole and navigate with the aid of GPS and visual detection methods to retrieve the ball and deposit it in the hole. The project was broken into four major subsystems: Communication, Navigation, Chassis, and Ball & Hole Locator. This paper discusses the details relating to the Ball and Hole Location Subsystem. Details include hardware and software design, theory of operation, interfacing, and associated design costs.

Table of Contents

1. Project Breakdown……………………………….………5
 1.1. Total System…………………….……….…..….5
 1.2. Ball and Hole Location Module……………….…..6
2. System Overview…………………..……………..8
 2.1. System Requirements & Design Goals…………..…8
 2.2. Subsystem Functions………………..………...9
  2.2.1. Camera……………………..……………………9
  2.2.2. Camera Interface……………………..…….…...10
  2.2.3. Camera Motion Control……………………….12
  2.2.4. Interfacing with the Navigation Subsystem………...13
  2.2.5. User Interface…………………..…………15
 2.3. Subsystem Integration…………………..……...16
  2.3.1. Power Distribution……………………..……….16
  2.3.2. Mechanical Layout……………………..……....17
 2.4. Software Design…………………..…………....19
  2.4.1. Algorithm Overview…………………….……..19
  2.4.2. Camera Communications………………………..20
  2.4.3. Searching……………………..………………...22
  2.4.4. Tracking……………………..………………28
  2.4.5. Chassis Control……………………..………….30
3. Resource Allocation…………………..………..32
 3.1. Power Distribution…………………..………..32
 3.2. Cost Summary………………………..…………..33
 3.3. Division of Manpower …………………..…..34
4. Testing…………………..……………………34
 4.1. Conditions…………………..…………………34
 4.2. Tests performed…………………..…………35
 4.3. Results…………………..……………………35
5. Conclusion……………………..………………..37
6. List of Tables……………………..……………....4
7. List of Figures……………………..…….…..4
8. List of Appendices ………………………………4

List of Figures

Figure 1: Project Breakdown…………………………………………......…..5
Figure 2: Ball/Hole Locator Block Diagram……………….……...…..7
Figure 3: CMU Camera and CMOS Image Sensor…………..…...…..8
Figure 4: Setup of the DUART………………..……………………..…...11
Figure 5: Modified 68HC12 Memory Map……………..……...…..12
Figure 6: Timing Diagram for Slave Out Data Transfer………..…..13
Figure 7: Timing Diagram for Master Out Data Transfer……...…..14
Figure 8: Power Distribution Diagram……………..……………..16
Figure 9: Component Layout……………..……………………..…..18
Figure 10: Flow Diagram for Scanning……………..……………..19
Figure 11: Flow Diagram for Camera Communication……………..22
Figure 12: Search Sector Layout……………..………………………..23
Figure 13: Repositioning Diagram……………..…………………..24
Figure 14: Flow Diagram to Find Target……………..……………..28
Figure 15: Flow Diagram for Camera Tracking……………......…..29
Figure 16: Data Packet Map……………..……………………………31

List of Tables

Table 1: Indicator Description……………..……………………….. 15
Table 2: Register summary for a using single channel DUART …..…20
Table 3: Summary of Commands for Color Tracking ……………..…27
Table 4: Data Packet Description…………………………………....31
Table 5: Power Budget…………………………………………….…32

Appendices

Appendix A: Serial Communication Schematics and Altera Code……….38
Appendix B: Mechanical Drawings and Pictures……………………..…43
Appendix C: Source Code for the MC68HC12……………………………45
Appendix D: Schematics…………………………………………….….60
Appendix E: Printed Circuit Board Artwork……………………..…..61
Appendix F: Project Costs………………………………………….…62

1. Project Breakdown

The following section is broken into two parts. The first involves the robot as a fully integrated system. A short definition of the role of each module is given. The second part of this section goes into detail of the Ball/Hole Location Module.

1.1. Total System

The robot is composed of four key parts each with their own primary functions. Figure 1 gives a diagram of how each part of the robot fits:

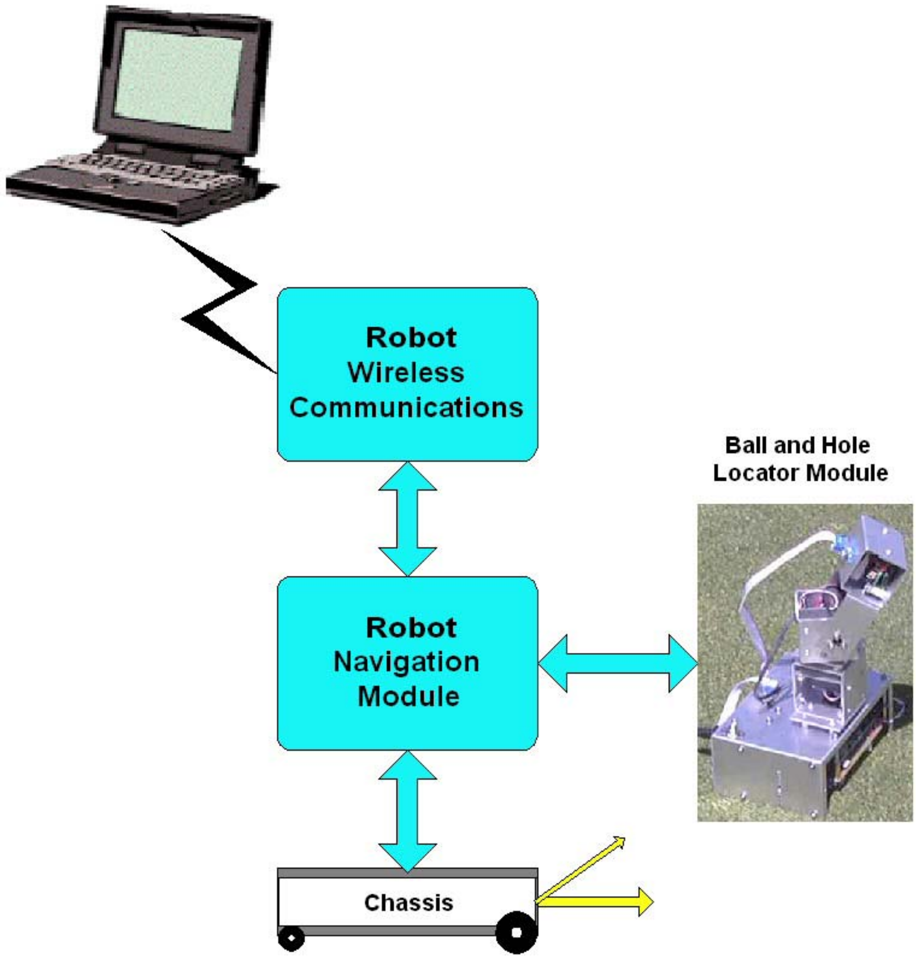


Figure 1: Project Breakdown

- Chassis-The purpose of the chassis is move a direction and distance specified by navigation in addition to capturing the ball or dropping the ball in the hole. The chassis is also responsible for supplying power to all the other modules.
- Communication-The primary purpose of the Communication Module is to send wireless data (GPS coordinates) to the Navigation module for processing. Communications transmits over a wireless link data that gives the status (position, speed, direction) of the robot to a remote base station.
- Navigation- Navigation's primary purpose is to calculate the direction and distance to travel using GPS guidance. Navigation is also responsible for controlling all of the inter-module communications on the robot.
- Ball/Hole Location- The Ball Hole Locator Module is responsible for finding the golf ball or hole once navigation is as close as GPS can get. The module scans for the desired object, and then tells the chassis where to go so that the ball can be picked up or dropped into the hole.

1.2. Ball and Hole Location Module

The general structure of this module is shown in Figure 2 as a block diagram. The module consists of four pieces that combine to make the Ball/Hole Locator Subsystem. The MC68HC12 microprocessor is the central part of the subsystem which controls the flow of information linking all individual systems together.

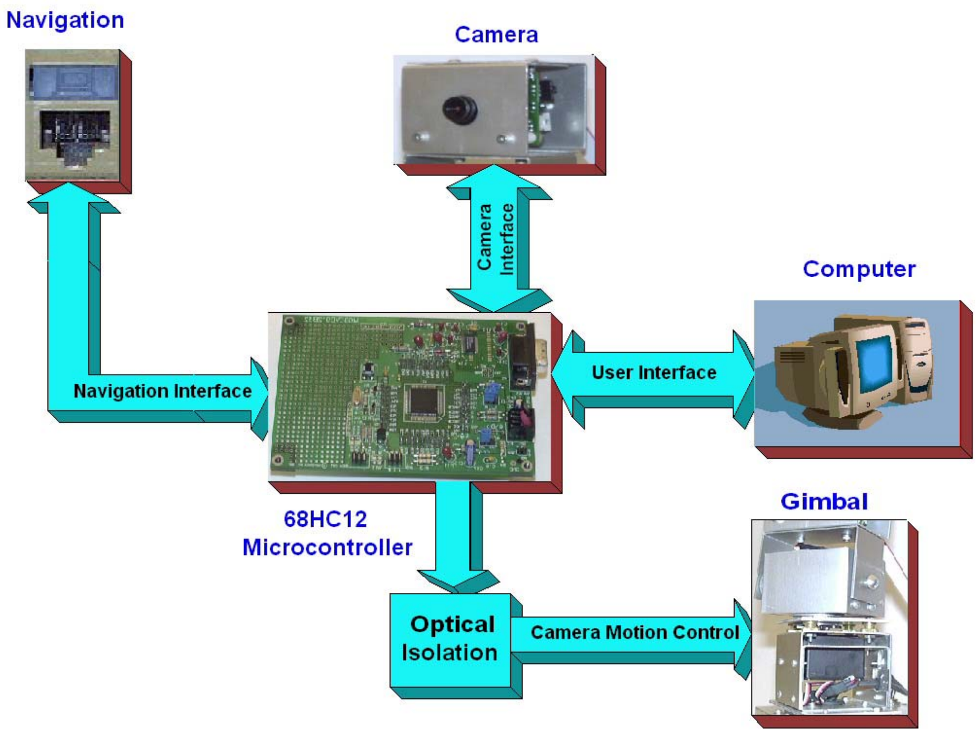


Figure 2: The Ball/Hole Locator Block Diagram.

The camera is the focal point of the design. It provides image data including target coordinates and mean color data. A serial interface was implemented so that camera commands could be sent to the camera and picture data could be received by the 68HC12 for processing. The microcontroller analyzes this data to determine which way to move the camera to keep the target centered in the field of view.

At the same time that the camera is constantly being repositioned, data is being generated and packaged by the 68HC12 so that it can be passed to the chassis via an interface to navigation. This allows the chassis to be "guided" to the final position of the ball or hole. The design also includes a user interface consisting of eight LED indicators and a level shifted serial port, which allows for convenient monitoring of the internal systems while the module is active.

2. System Overview

The following section outlines the design of the Ball and Hole Locator Module. Details regarding system requirements, hardware, and software design.

2.1. System Requirements & Design Goals

The specified task for the Ball and Hole Locator was to pinpoint the golf ball or hole visually once navigation gave control of the robot to the ball and hole locator subsystem. The navigation subsystem was able to get within 3 meters of the ball or hole assuming that the GPS (Global Positioning System) receiver was receiving a WAAS (Wide Area Augmentation System) signal. Additional size limitations were imposed by the chassis. In addition to the requirements defined by the problem and limitations set by the technology a list of design goals were defined to meet in addition to the basic system requirements. The following is a summary of environmental conditions as well as these requirements and goals.

Environmental Conditions

- The horizon is not controlled and may contain buildings and various vegetation
- The ball would be on green grass (not dying)
- Testing would be conducted from 11 a.m. until 5 p.m. in various weather conditions.

System requirements

- Upon request from navigation, find a Ball or Hole within a 3 meter radius.
- The golf ball will be and unmodified standard white golf ball.
- The hole dimensions will follow the guidelines set by the PGA.
- The hole will be marked using a roll of paper inside of the hole that stands 3cm above the plane of the grass. The diameter of the roll will be equal to the diameter of the hole itself.

- Once the ball or hole is found, Send commands the chassis to guide it to the ball or hole.
- Sensors must work in outdoor conditions.
- All electronics must fit within a 3”x5”x7” aluminum enclosure.
- Voltages available from chassis include 5,6, and 12 Volts DC.

Design Goals

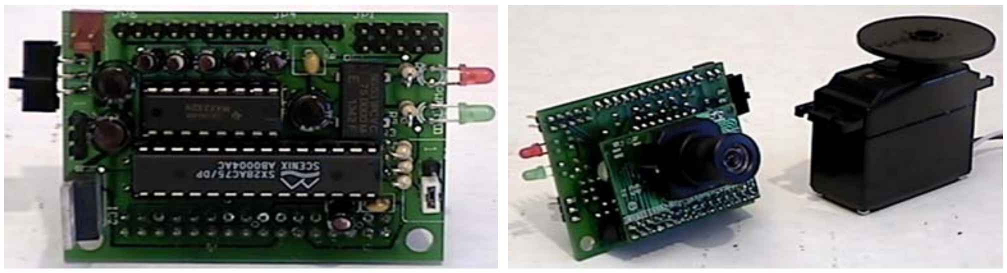
- Simple interface to robot: One power plug and one data plug.
- Self contained modular design.
- Utilize printed circuit board technology whenever possible.
- Use indexed locking wire connections.
- Keep all wire bundled in harnesses whenever possible.
- Be resistant to noisy or faulty power.
- Use optical isolation on any motors or servos.
- Keep design easy to fix, test, and debug.
- Make all parts physically robust, reliable, and easy to disassemble.

2.2. Subsystem Functions

2.2.1. The Camera

The CMOS sensor chosen for this project was a pre-built camera with several features. Among them, the most relevant include

- *Ability to track a defined color at 17 frames per second.*
- *80x143 Pixel Resolutions.*
- *Serial Communication Port.*
- *Outputs size data and center coordinates of an object.*
- *75ohm output for live video feed to a TV.*
- *Pre-programmed 75MHz Video Processor.*



Fig

Figure 3: CMU camera CMOS Image Sensor.

Features including the ones mentioned above made the CMU camera a feasible solution to the problem of image detection. The ability to communicate through the serial port made it ideal for use with the HC12 microprocessor. Also, a fast video processor did a much better job processing raw image data with the HC12. A Java GUI program included with the camera made testing and debugging simple. The outdoor testing helped to develop an acceptable routine for scanning and tracking. The CMOS camera was the most feasible solution to the problem of image detection.

2.2.2. Camera Interface

In order to control camera commands such as color tracking and auto gain, a Serial Communication Interface (SCI) was required. The camera has a serial output for both TTL/CMOS and level shifted data. Also, the camera baud rate can be changed between 38400 bps and 115000 bps by changing a jumper which can be found in the camera documentation. The 68HC12 does contain a SCI and a Serial Peripheral Interface (SPI). Since the choice was made to use the SPI for communication to navigation, a Serial port expansion for the 68HC12 was required.

Thus, the most widely used device for a serial interface is the Universal Asynchronous Receiver/Transmitter (UART). The UART used in this application is the National Semiconductor PC16552D. This part is a DUART which contains *dual* serial ports. The DUART was selected because of the extra serial port and adjustable baud rate. The DUART expansion was built according to the following design criteria:

- Two channel selection - the expansion should include both ports for easy of use.
- Level Shifted output - A level shifted output will assist in debugging codes and gives the ability to watch conversations between the 68HC12 and the camera.
- A simple hardware interface – one plug between the 68HC12 memory expansion and the UART.

The UART was wired wrapped in the prototype area of the 68HC12 evaluation board according to the diagram show below in Figure 4. An external oscillator of 18.432 MHz was used to generate the baud rates of 38400 bps or 115000 bps. Also, a MAX233 dual level shifter was built to provide a level shifted output for both channels.

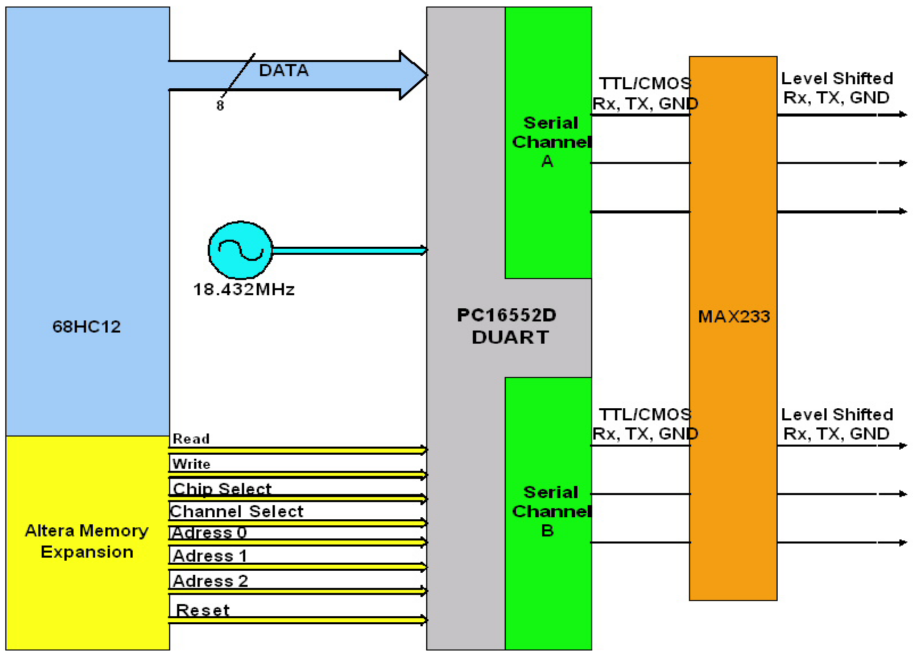
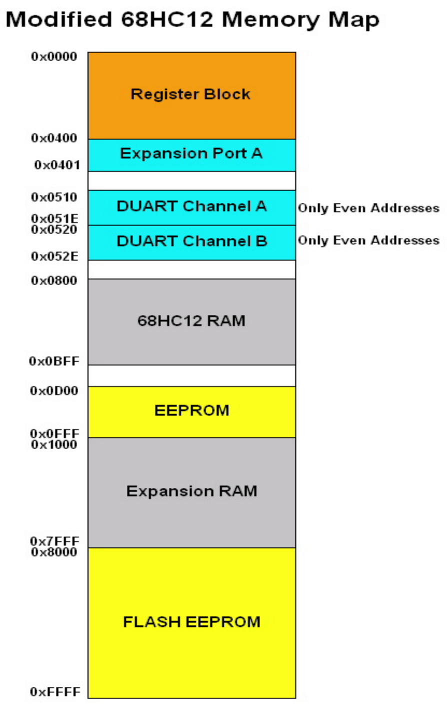


Figure 4: Setup of the DUART

The expansion required making three main changes to the Altera chip on the memory board. First, the memory map was changed to include room for the new registers as shown below in Figure 5. Only even address were used since the lower eight data bits were wired into the data pins of the DUART. Second, an address decoder was built to provide the chip select and channel select for the DUART. Finally, general I/O expansion port B was wiped out so that external pins such as reset, chip select, channel select, and read/write could be wired to the DUART. The final changes to the Altera code, including the address decoder, can be found in Appendix A, Figure 3**.**



After the Altera board was re-programmed, the parts were populated on the microcontroller and wire wrapped on the board. A cable was made so a simple connection could be made from the memory expansion to the DUART. A detailed schematic of connections and a picture highlighting the components can be found in Appendix A, Figure 2. The registers on both channels were checked by writing and reading from the appropriate memory locations. The divisor registers can be set to give either 38400 bps or 115000 bps. Also, the data ready bit was checked as a flag to determine when the UART received new data. For more information on how the DUART was programmed please refer to section 3.4.2.Camera Communications.

2.2.3. Camera Motion Control

The camera motion control system is used to keep the camera pointed at the target (ball or hole). This is done using an Elevation over Azimuth positioning system, or gimbal driven by two standard servos. The servos are controlled by the Pulse Width Modulation Subsystem (PWM) of the 68HC12. This PWM signal is fed from the HC12 through an Optical Isolator to the servos. The servos require a 50Hz signal with a pulse width of 0.7ms to 1.7ms for the full range of movement. To get this kind of resolution within a 20ms period, one 16-bit channel is used per servo. The result is that if the duty cycle is changed by a decimal value of 18, the servo moves 1 degree. It is important to note that the optical isolators invert the logic signal that they receive; as a result, positive polarity must be used when setting up the HC12 PWM subsystem. See Appendix B for detailed pictures of the gimbal.

2.2.4. Interfacing with the Navigation Subsystem

Communication with navigation is achieved using synchronous serial communication. Navigation controls the S Clock, Slave Select, and the Baud Rate used during data transfers.

The communications follows a transfer protocol as follows:

- The Navigation module is set up as a master and the Ball/Hole module set as a slave.
- Next, the MSTack line is raised in order to acknowledge that navigation has data that is ready to be sent.
- When the slave is ready to receive data, SLAack is brought high.
- Navigation then sends the data.
- The MSTack line is lowered, indicating that the data transfer is complete.

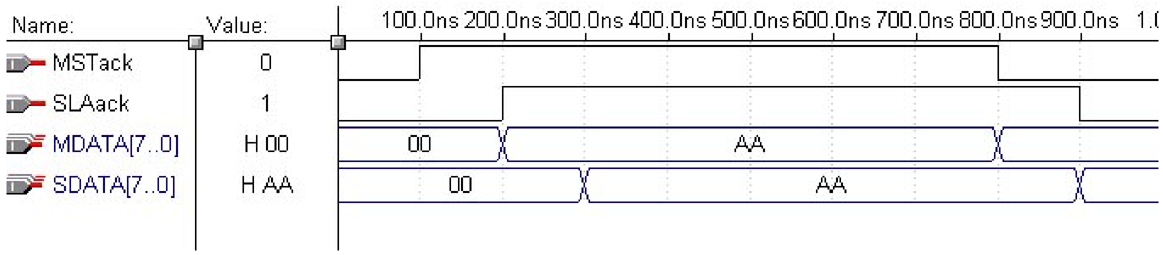


Figure 6: Timing Diagram for a Slave out Data Transfer.

Sending data to the navigation module follows a similar process:

- The Navigation module is set up as a master and the ball/hole module set as a slave.
- Next, the SLAack line is raised in order to acknowledge that Ball/Hole has data that is ready to be sent.
- When Navigation is ready to receive data, MSTack is brought high.
- Ball/Hole then sends the data.
- The SLAack line is lowered, indicating that the data transfer is complete.

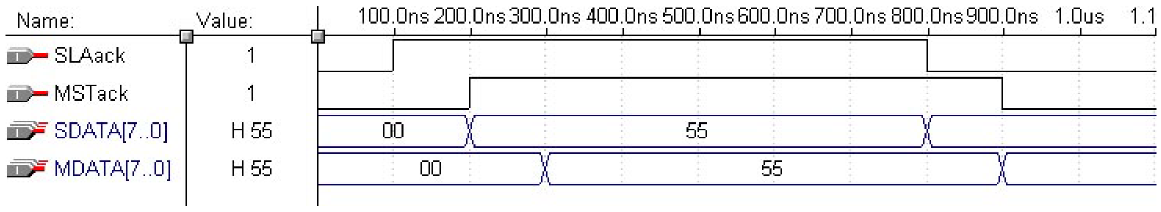


Figure 7: Timing Diagram for a Master out Data Transfer.

2.2.5. User Interface

To make the manner of troubleshooting the module easier, eight Light Emitting Diodes (LEDs) were added. The LEDs indicate the current state that the module is in. The current function would be indicated by the indicator LEDs. The eight specific functions shown by the LEDs are show below in Table 1. There are eight specific functions that are indicated by the LEDs.

Indicator	***Name***	***Description***
1 (Top Indicator)	**Reset Camera**	**Hard Reset on camera has occurred**
2	**Correct Gimbal**	**Camera is correcting the gimbal**
3	**Scan**	**Module is currently scanning**
4	**Send**	**Module is sending data to navigation**
5	**Wait**	**Module is waiting for a trigger**
6	**Lock**	**Object of interest has been found**
7	**Send to Camera**	**Command is being to sent to camera for processing**
8 (Bottom Indicator)	**Receive from Camera**	**Data is being received from camera**

Table 1: Indicator Description

2.3. Subsystem Integration

2.3.1. Power Distribution

Power is supplied by the chassis. The Ball/Hole Locator Module requires two supply voltages. A 5 volt supply is used by all the electronics in the system. The 6 volt supply is used to power the servo motors. This was done to isolate any noise generated by the servos from the sensitive electronics.

Power is fed through a fuse, through the main power switch, and on to the power distribution board. The Power distribution board provides a place for all of the internal components to be connected to power. The board filters the 5 volt power with a 100uf capacitor and the 6 volt supply is filtered by a 3300uf capacitor. Figure 8 shows the flow of power to each subsystem.

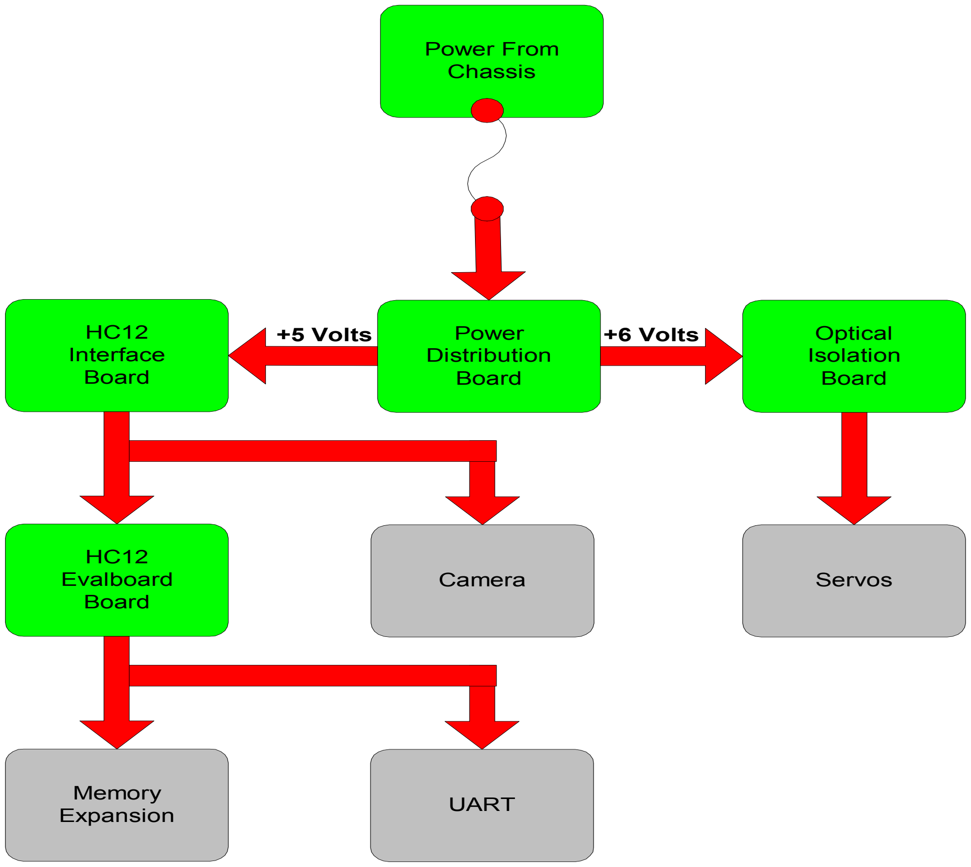


Figure 8: Power Distribution Diagram

2.3.2. Mechanical Layout

The specifications defined by chassis to fit all the electronics within a 3”x5”x7” aluminum enclosure is the main limiting factor for the layout of the electronics package. The exception to this space limitation is the camera and the camera positioning system. The different components within the electronics package are laid out to minimize the amount of “spaghetti wire” in addition to keeping the wires used for inter-device connections to a minimum.

The HC12 interface board is used to connect the HC12 Evaluation board to all other components in the electronic package excluding the UART. The UART is built onto the Evaluation board in the prototype area as well as being connected by a 10 pin ribbon cable to the memory expansion board. Using an interface board is beneficial for several reasons. First, it allows other components to be connected and disconnected to the HC12 all at once, as well as preventing improper connections. It also allows the use of a variety of different types of connectors, so that no plugs can be plugged into the wrong place or with the wrong polarity. Finally, the board provides space for trivial circuitry such as pull down resistors and noise filters.

Below is a diagram (figure 9) that illustrates the layout of all the major components of the Ball and Hole Locator. For detailed explanations of what each subsystem does refer to section 3.2 Subsystem Design.

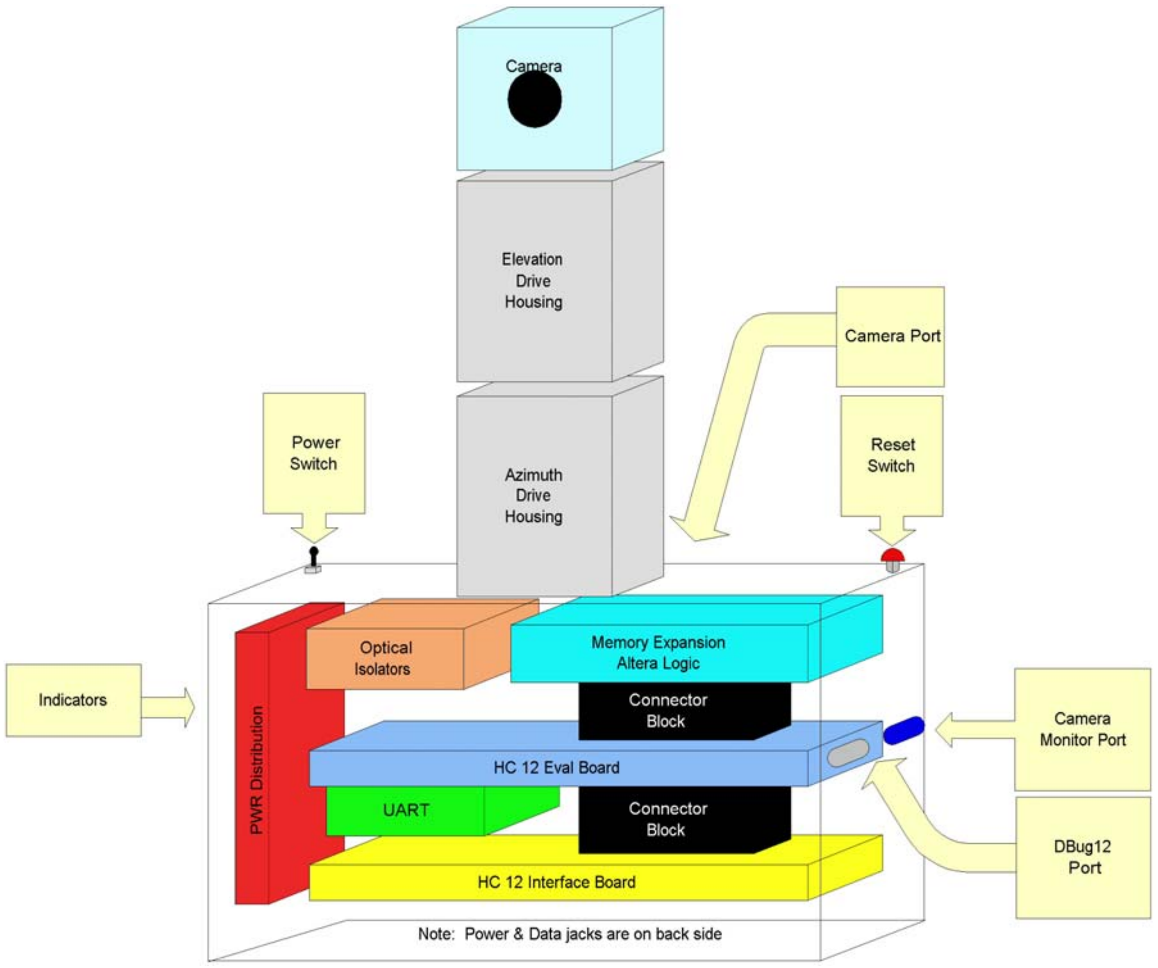


Figure 9: Component Layout

2.4. Software Design

2.4.1. Algorithm Overview

The scanning algorithm follows the process shown below in Figure 10. The scan starts by waiting for a request to find the ball or the hole, then the scan moves to an initial sector and looks for any bright objects. If the camera finds a bright object with a specific size and enough confidence, then it "locks" unto that target. The camera will keep moving to a new sector when there is "no lock" in the current sector. If the camera completed the scan, it sends a command requesting the robot to reposition and the scan is run again. When a "lock" is made the position of the object is found and the camera is moved to directly at the target. The same "lock" process is used while tracking to verify that the target has not been lost. When the object is found the program will go back into a "wait" state until another request is made. Any time the lock becomes lost during the camera correction process, then the object is assumed to be "lost" and the scan restarts.

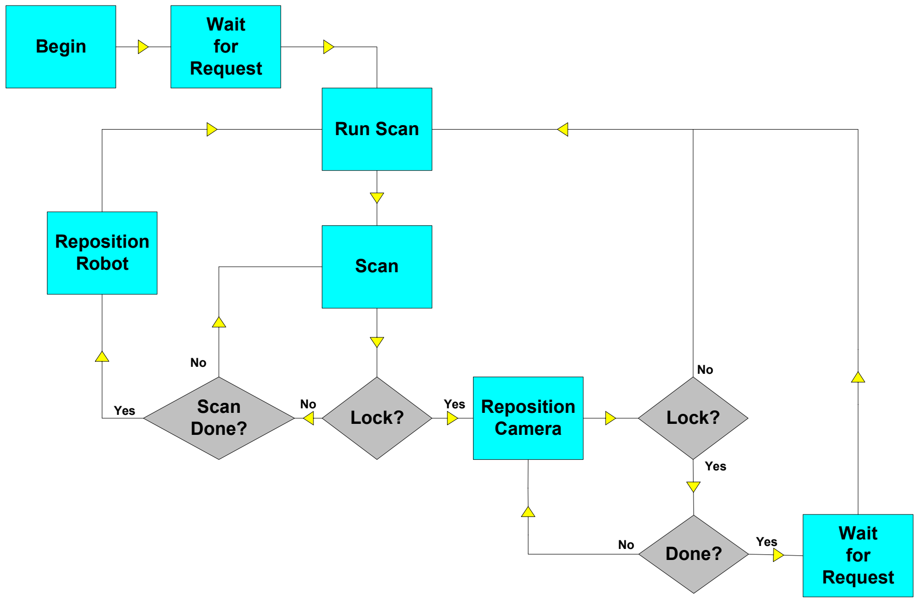
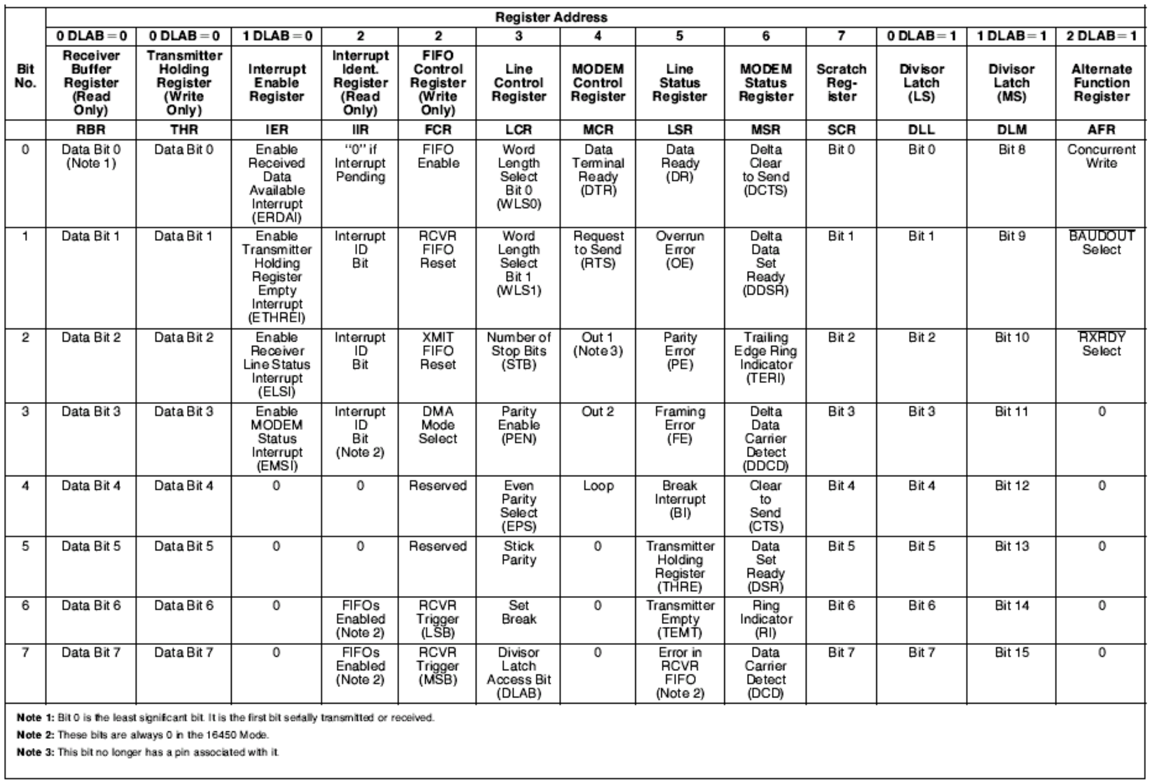


Figure 10: Flow Diagram for Scanning.

2.4.2. Camera Communications

The camera communication software is designed for use with a National Semiconductor DUART (**See 3.2.2. Camera Interface**). Both channels have eight registers through which setup, transmitting, and receiving are controlled. These registers are defined in Table 2.

Table 2: Register summary for a using single channel DUART.



Bit No.	Register Address												
	0 DLAB = 0	0 DLAB = 0	1 DLAB = 0	2	2	3	4	5	6	7	0 DLAB = 1	1 DLAB = 1	2 DLAB = 1
	Receiver Buffer Register (Read Only)	**Transmitter Holding Register (Write Only)**	**Interrupt Enable Register**	**Interrupt Ident. Register (Read Only)**	**FIFO Control Register (Write Only)**	**Line Control Register**	**MODEM Control Register**	**Line Status Register**	**MODEM Status Register**	**Scratch Reg-ister**	**Divisor Latch (LS)**	**Divisor Latch (MS)**	**Alternate Function Register**
	RBR	**THR**	**IER**	**IIR**	**FCR**	**LCR**	**MCR**	**LSR**	**MSR**	**SCR**	**DLL**	**DLM**	**AFR**
0	Data Bit 0 (Note 1)	Data Bit 0	Enable Received Data Available Interrupt (ERDAI)	"0" if Interrupt Pending	FIFO Enable	Word Length Select Bit 0 (WLS0)	Data Terminal Ready (DTR)	Data Ready (DR)	Delta Clear to Send (DCTS)	Bit 0	Bit 0	Bit 8	Concurrent Write
1	Data Bit 1	Data Bit 1	Enable Transmitter Holding Register Empty Interrupt (ETHREI)	Interrupt ID Bit	RCVR FIFO Reset	Word Length Select Bit 1 (WLS1)	Request to Send (RTS)	Overrun Error (OE)	Delta Data Set Ready (DDSR)	Bit 1	Bit 1	Bit 9	$\overline{\text{BAUDOUT}}$ Select
2	Data Bit 2	Data Bit 2	Enable Receiver Line Status Interrupt (ELSI)	Interrupt ID Bit	XMIT FIFO Reset	Number of Stop Bits (STB)	Out 1 (Note 3)	Parity Error (PE)	Trailing Edge Ring Indicator (TERI)	Bit 2	Bit 2	Bit 10	$\overline{\text{RXRDY}}$ Select
3	Data Bit 3	Data Bit 3	Enable MODEM Status Interrupt (EMSI)	Interrupt ID Bit (Note 2)	DMA Mode Select	Parity Enable (PEN)	Out 2	Framing Error (FE)	Delta Data Carrier Detect (DDCD)	Bit 3	Bit 3	Bit 11	0
4	Data Bit 4	Data Bit 4	0	0	Reserved	Even Parity Select (EPS)	Loop	Break Interrupt (BI)	Clear to Send (CTS)	Bit 4	Bit 4	Bit 12	0
5	Data Bit 5	Data Bit 5	0	0	Reserved	Stick Parity	0	Transmitter Holding Register (THRE)	Data Set Ready (DSR)	Bit 5	Bit 5	Bit 13	0
6	Data Bit 6	Data Bit 6	0	FIFOs Enabled (Note 2)	RCVR Trigger (LSB)	Set Break	0	Transmitter Empty (TEMT)	Ring Indicator (RI)	Bit 6	Bit 6	Bit 14	0
7	Data Bit 7	Data Bit 7	0	FIFOs Enabled (Note 2)	RCVR Trigger (MSB)	Divisor Latch Access Bit (DLAB)	0	Error in RCVR FIFO (Note 2)	Data Carrier Detect (DCD)	Bit 7	Bit 7	Bit 15	0

Note 1: Bit 0 is the least significant bit. It is the first bit serially transmitted or received.
Note 2: These bits are always 0 in the 16450 Mode.
Note 3: This bit no longer has a pin associated with it.

Setup of the DUART for communication requires that several registers are set correctly. The camera requires a baud rate of 38400 or 115200 bps with no flow control, one start bit, one stop bit, and no parity. In the main program (See Appendix C,) DLAB was enabled in the line control register so that the baud rate can be set. Depending on the clock source used, the correct values must be set in the divisor registers (for 18.432MHz, a value of decimal 30 was written in the divisor registers to give a baud rate of 38400 bps). Then concurrent write was turned off[1]. After the baud rate was set, DLAB was cleared so the Receive/Transmit register can be used. Interrupts were not needed for communication. The FIFO buffer was enabled in the FIFO control register. The interrupt enable address was checked to confirm that no interrupts or data was enabled. When new data is ready for reading from the receive register the receive line status bit is set. The register is used as a method for checking for new data.

Once the DUART was set up, two functions had to be written: send commands and receive data. The code used in the main program to send a specific command to the camera needed to be as painless as possible. The command function was the result of this requirement (see Appendix C). Command works by receiving a string of a variable length. This string is then clocked out, one character at a time, into the Receive/Transmit register. This continues until the function reaches the end of string marker. The crucial step was to add just enough of a delay so that the UART doesn't write too fast, hence missing a key part of the string. Therefore, a small 5 ms delay was added between successive writes to the receive/transmit address. The final format for calling the function was: **command** (*string*); where *string* can be any length or a variable. There are two methods implemented to help trouble shoot the command function. First, an optional printf() command was used to print each character as it was clocked out. Also, since the port was level-shifted, the string could be watched separately from another computer setup as an "observer".

For the receiving program, assuring all data arrives at the correct time was the hardest part. The receiving program was the solution to this problem (see Appendix C). The receive function uses a string of twenty-five characters named *camdat* when called. The parameter passed to this function is the number of characters to be received. In order to get the data at the correct time, the interrupt enable register is checked to see if the receive line status bit is set. If the flag went high, then data is ready and is immediately read into *camdat*. This

[1] NOTE: the DUART *will not operate correctly* if concurrent write is enabled. Also, when disabling concurrent write the divisor registers or DLAB may reset. Therefore, it is recommended that both divisor registers and alternate function registers continually be written to until set correctly.

happens until all required characters are received. Thus, the format for this function call is **receive**(*n*), where n is the number of bytes. Figure 11 shows the flow diagrams of both the command and receive functions.

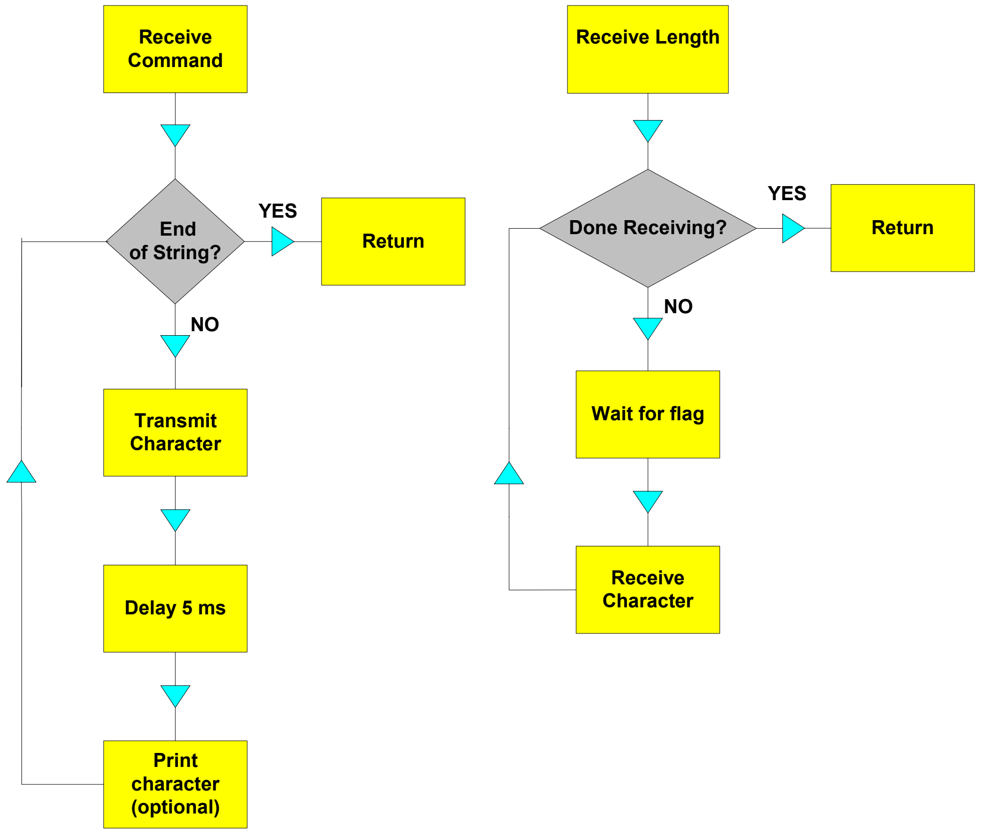


Figure 11: Flow Diagram for Camera Communication

2.4.3. Searching

In order for the camera do a successful search, the camera must be able to see the entire search area one way or another as well as having a routine to "look" for a target (ball or hole) within each "frame" or field of view. The first half of this section will describe how the camera is repositioned from one search area to the next. The second half of this section will describe the process for finding a target within each sector.

Repositioning the Camera

Searching is done by one of two ways. The first is using the gimbal to reposition the camera. The gimbal allows the camera to be repositioned anywhere from -60 degrees to +60 degrees azimuth and 0 to 90 degrees elevation. This combination allows the camera to see from 0.3m in front of the robot to beyond the visible range of the camera (>2.25m). With the combination of these 2 degrees of freedom, the camera can see a full 120 degrees of the horizon near and far. This area is divided into 10 different sectors that the camera searches. Each sector boundary has 6 degrees of overlap to ensure full coverage. These Sectors are shown below in figure 12, the numbers in each area represent the search order.

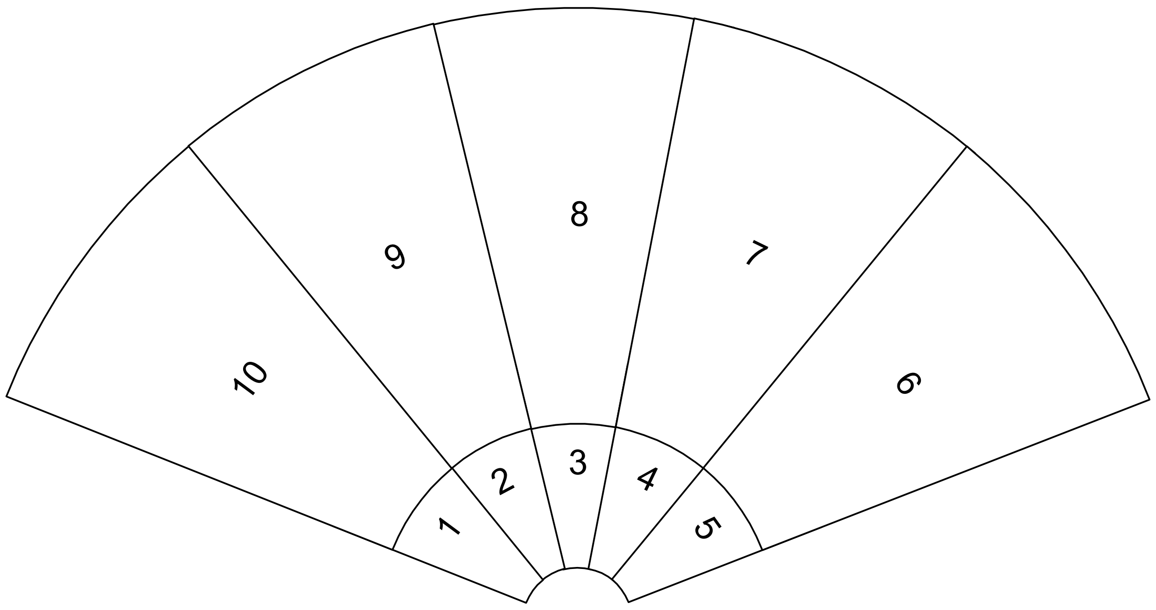


Figure 12: Search Sector Layout

Since these 10 sectors cover only 120 degrees of the horizon, it is necessary to combine these search areas into a larger pattern that the chassis makes possible by repositioning the robot. Each time the camera searches all 10 sectors without a lock, a "reposition" command is sent to the chassis. The first two times the command is sent, the robot will rotate 120 degrees and wait for us to do another search. This allows the camera to cover the full 360 degrees of the horizon. The downside of this is that there is a blind spot directly under the robot and immediately around it. To cover this area, the chassis goes in reverse 1m after receiving the third “reposition” command thus allowing the camera to directly see the remaining area. Figure 13 shows how the entire search area is broken down into smaller areas that can be searched using the pattern shown in Figure 12.

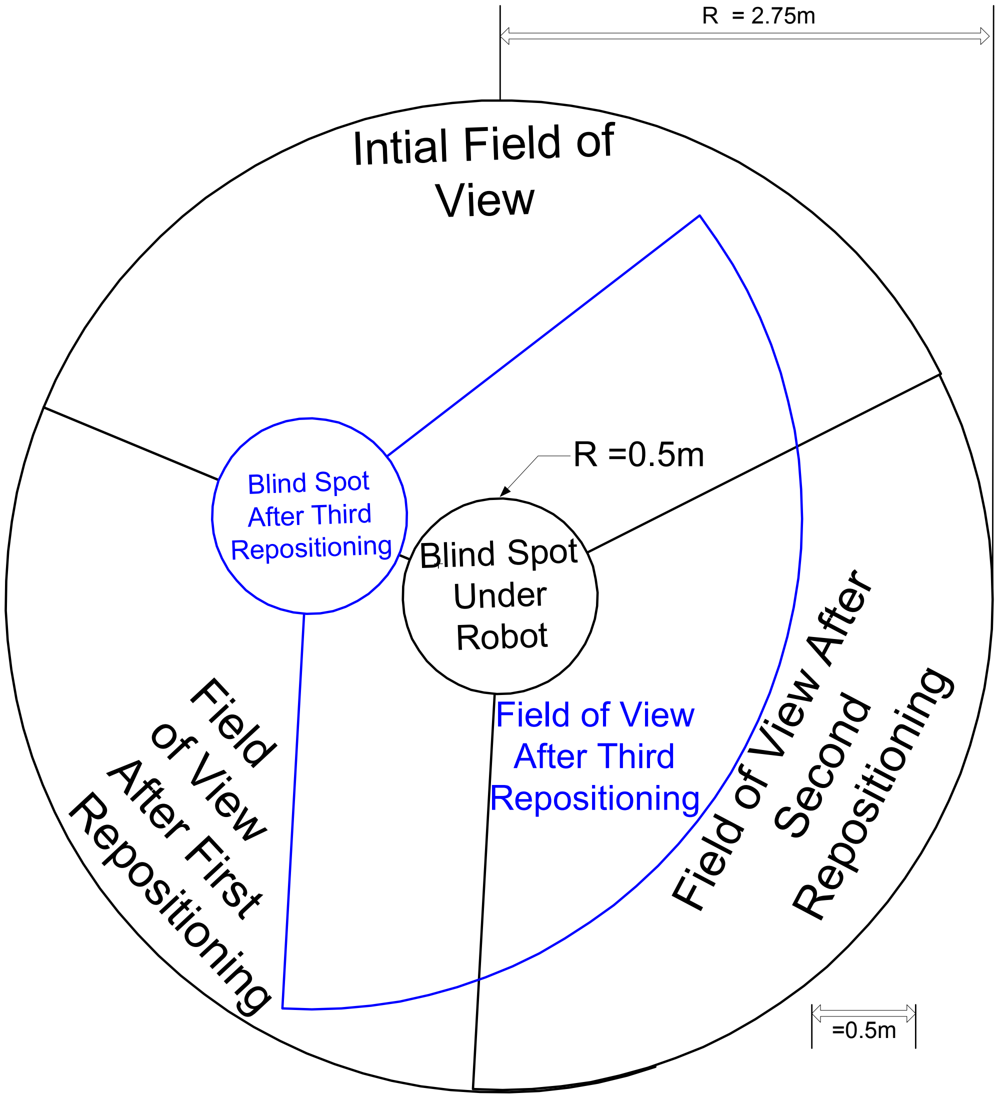

R = 2.75m
Intial Field of View
R =0.5m
Blind Spot After Third Repositioning
Blind Spot Under Robot
Field of View After First Repositioning
Field of View After Third Repositioning
Field of View After Second Repositioning
=0.5m

Figure 13: Repositioning Diagram

This method allows for a complete search within a 2.75 meter radius. Navigation guarantees less than a 3 meter error in position. This leaves a small chance that the ball is beyond the range of the camera. To deal with this, plans were made to have the chassis navigate the circumference of the search perimeter stopping every 2 meters to perform a scan. This was never implemented by the chassis but it is already built into the code for the ball and hole location subsystem.

Finding the Ball and Hole

Once the camera is looking in a certain direction it must look for the ball or hole. This is done by sending a command to the camera called “get mean.” The camera returns a string of data consisting of the mean values of the red, green, and blue channels, in addition to the variance of each channel. This data is used to calculate the minimum color values to be tracked. The minimum color value is found using the equation: {MinColor = MeanColor + 2*Variance}. This is done only for the green and blue channels. The minimum value for red is set to 0. This is done to ignore the red part of the spectrum. By ignoring red, the camera is not distracted by the horizon or the sky since red varies greatly depending on whether the camera is looking at all grass or if the horizon/sky is in the frame.

The maximum values of red, green, and blue to be tracked are always 240 (saturation) since white objects are the only thing that can reflect enough light to saturate the red, green, and blue channels, and as a result, it makes them easy for the camera to track.

Small white objects as well as large distant white objects that have similar angular size as a golf ball will confuse the camera since they resemble a ball and/or hole. To prevent the camera from tracking objects that are too big to be either a ball or hole, the size is checked to make sure that the object is smaller than 50 pixels.

Once the minimum values of green and blue are calculated they are put into a command string that is sent to the camera. The string is in the form:

“TC Red_{min} Red_{max} $Green_{min}$ $Green_{max}$ $Blue_{min}$ $Blue_{max}$ \r”

The camera returns a string in the form:

“255 M M_x M_y X_1 Y_1 X_2 Y_2 Conf Size:”

“255 M” is a packet identifier. “Conf” refers to the confidence that the camera has about what it is tracking, any value above 50 is considered a definite lock. “Size” refers to the size of the target in terms of pixels. Any value greater than 50 cannot be a ball or hole. “M_x M_y” will be discussed in section 3.4.4 Tracking. ” X_1 Y_1 X_2 Y_2” are not used. For more information on the commands sent to the camera as well as the format of the data packets generated by the camera refer to Table 3 on the following page.

Table 3: Summary of Commands for Color Tracking

Command	Command String Format	Return Packet Format
Get Mean	**"GM\r"**	**"255 S R_m G_m B_m R_v G_v B_v :"** **S** is the type of packet returned ($_m$) represents the mean value for a single .channel ($_v$) is the variance returned for a single channel.
Track Color	**"TC R_{min} R_{max} G_{min} G_{max} B_{min} B_{max} \r"**	**"255 M M_x M_y X_1 Y_1 X_2 Y_2 C_{onf} S_{ize} :"** **M** is the type of packet returned. **M_x** This is the x-coordinate of the center of the object tracked. **M_y** This is the y-coordinate of the center of the object tracked. **X_1 ,Y_1** is the lower left coordinate of the target. **X_2 ,Y_2** is the upper right coordinate of the target. **C_{onf}** This is the confidence of the tracked object. **S_{ize}** This is the size of the tracked object

Below is a block diagram that shows the steps necessary to find the ball or hole within a sector. (Figure 14)

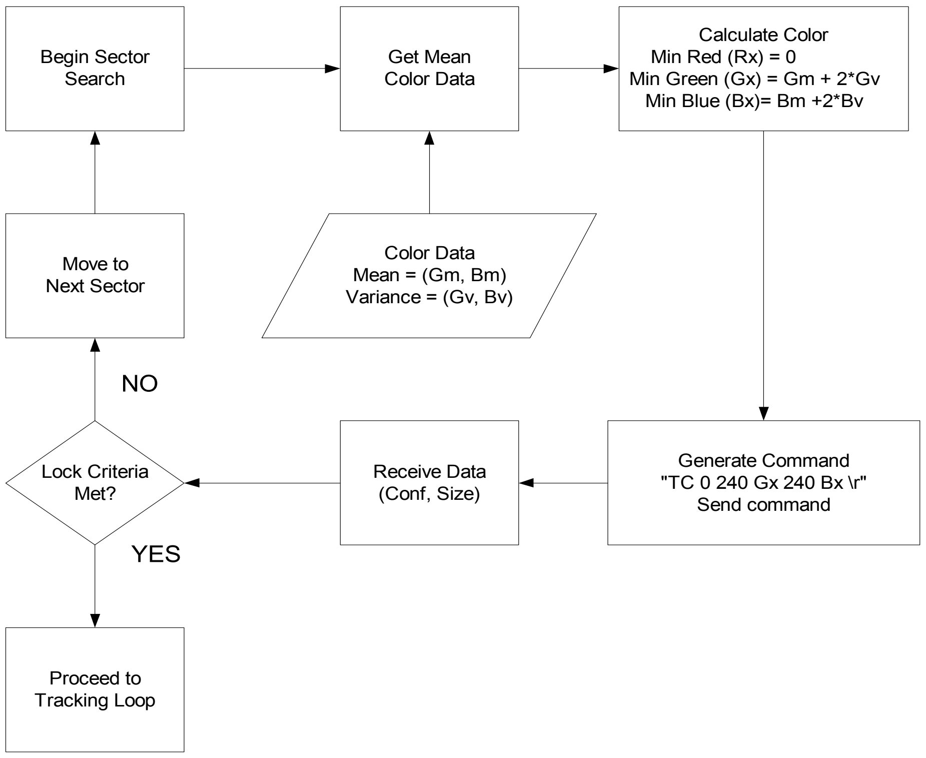


Figure 14: Flow Diagram to Find a Target

2.4.4. Tracking

Tracking is initiated by sending a "track color" command to the camera. The command is sent with a set of six arguments that define what to look for. The arguments include the minimum and maximum values of the red, green, and blue components of the target color. If the camera sees an object that meets the color criteria and size criteria a visual lock is made.

When a target is visually locked, the coordinates of the target is loaded from the camera into the 68HC12 memory. These coordinates represent the position of the target within the field of view of the camera, in an X-Y plane. The X values (horizontal) range from 0-80. The Y values (vertical) range from 0-143. The coordinates of the center field of view are (40, 72). The program takes the coordinates the target (Mx, My) and subtracts the coordinates of the center field of view. The difference between these numbers represents the angular distance the camera should move to center the target within the field of view. This correction is a signed value that is scaled or multiplied by a constant and added to the duty cycle of the respective servo.

This process allows for very fast tracking since the camera position is fully corrected on every program cycle. Figure 15 shows a block diagram of how this function works.

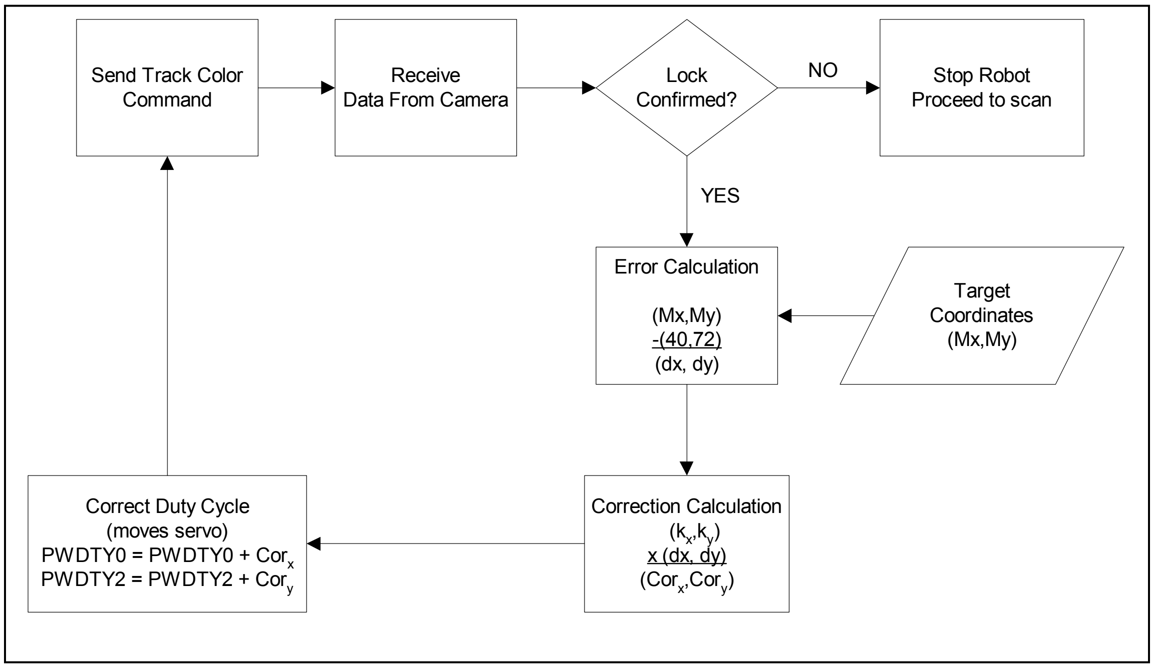


Figure 15: Flow Diagram for Camera Tracking

2.4.5. Chassis Control

While the Ball/Hole Subsystem is in a tracking loop, the chassis must be given instructions that tell the robot to stop, go, turn, or go strait every 260ms. This accomplished by sending a command every fourth RTI Interrupt. For this to be done, the data must be generated and packaged into one byte packets that contain a 2 bit identifier as well as data for the chassis to use. The first half of this section describes how the data is extracted and packaged into one byte packets. The latter half describes what kind of packet is sent each time data is sent.

Generating Control Data

In order to tell the chassis where to go so that the appropriate action with the ball can be taken, the module needed to send out drive and steer data to the chassis. The data that is to be sent consists of an 8-bit word. The 8-bit word is broken down into three distinct sections:

- CH4-CH0 bits that determine the magnitude of the drive or steer.
- CH5 high tells the chassis to either go backward, or turn left, depending on what the CH6 and CH7.
- CH6 and CH7 tell the chassis to drive, steer, or stop. If the 2 most significant bits are “0F”, the chassis will stop. If the command of a “01” is sent, the chassis will be told to steer. Finally, if a “10” is sent, then the chassis will be ordered to drive. CH6 and CH7 also tell the robot to start looking for the ball or hole, to reposition, or to tell the robot that the module is finished. See Figure 16 and Table 4 for more details.

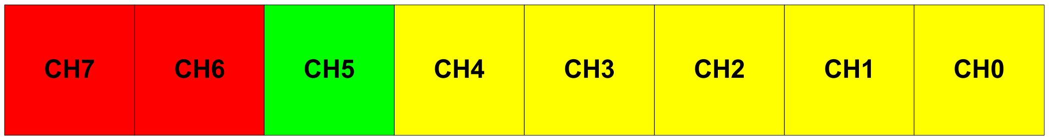


Figure 16: Data Packet Map

CH7	CH6	CH5	*CH4*	*CH3*	*CH2*	*CH1*	*CH0*	*Description*
0	0	0	*0*	*1*	*1*	*1*	*1*	*Stop*
0	1	0	*X*	*X*	*X*	*X*	*X*	*Steer to the Right*
0	1	1	X	X	X	X	X	Steer to the Left
1	0	0	X	X	X	X	X	Drive Forward
1	0	1	X	X	X	X	X	Drive Backward
0	0	1	0	0	0	0	0	Acknowledge to find the hole
0	0	1	0	0	0	0	1	Acknowledge to find the ball
1	1	1	1	0	0	0	0	Reposition the Chassis
1	1	1	1	1	1	1	1	Task Complete

Table 4: Data Packet Description

Transmitting the Data

When the target is initially acquired, there is a chance that the target is relatively close to the robot, but at a wide angle. This situation may result in the robot driving past the ball because the steering was too slow with respect to the driving speed. To prevent this scenario, a test is done to see if the ball is more than 10 degrees from the direction the chassis is pointing. If the chassis is too far out of line, only steering packets are sent until the chassis is pointing within the 10 degree range of the target. After the robot is pointing in the right direction the data stream consists of alternating packets (steering or driving). This is done until the target is in position for retrieval at which time a code (0xFF) is sent to signal that the Ball/Hole Locator is finished. If the target lock is lost during the tracking routine, the stop code (0xF0) is sent immediately to stop the chassis so that the target can be relocated.

3. Resource Allocation

This section will describe in detail about the power usage of the Ball/Hole subsystem with details of how much power each of the major components consume. The cost summary will cover how the money was spent, as well as any donations that were given. Finally, the section will end with the distribution of tasks among the team.

3.1. Power Distribution

Power requirements were determined previously by the amount of power each primary component will consume. All calculations were based on a 5-volt regulated DC power supply. Primary components and their power consumption include:

Components	Power Consumed
CMUcam	**100mW**
HC12	**300mW**
2 Futaba S3401 Servo Motors	**200mW**
HC12 Altera Expansion	**350mW**
Miscellaneous Discrete Components	**100mW**
Total Power Consumed	**1.05 W**

Table 5: Power Budget

Total power consumption is about 1 watt, which was adequately supplied by a regulated 5-volt power supply. These components continuously consume power, so a steady supply of power was necessary to keep the module up and running. More components could have been added; such as a voltage regulator to reduce the amount of power consumed, but the power drop wouldn't have been that significant. Also, if the expansion didn't consume as much power as it should, then the power requirement could have been further reduced and we wouldn't have to consort to continuously swapping out batteries for every half- hour of usage.

3.2. Cost Summary

Out of the $2,000 proposed for each robot, $600 was allocated for the team to utilize. shows the types of items that were purchased, the unit cost, how many units of the product were purchased, and the total cost of the product. Comments were also added in the table to indicate the source of the purchases. If a replacement parts are needed, info about the place where the product was purchased was included

As shown in the final budget, the total cost for all the components come to a total of $448.63. This includes all of the replacement parts that needed to be purchased as well as the shipping and handling costs. Miscellaneous costs that were useful in the design process are also included in this budget. For example, in the final budget, the 6' x 6' Astroturf was useful in testing basic tracking functionality of the camera.

Costs would have been further reduced if:

- Less failures in servo motors
- Excess material
- Ordering spare parts

However, components did fail and some purchases needed to be made to compensate those failures. For example, the first pair of the Futaba servos started to act very unreliably about three weeks before the final design review took place. Hence, two new servos had to be ordered, which tacked on another $100 to the budget. Due to multiple mistakes in the board design process, major setbacks were encountered building the power distribution and optical isolator boards. The result of these mistakes caused the current supply of the blank PCB to be used completely, and another blank PCB board had to be purchased for $8.50. All in all, an extra $108.50 was spent on replacement products. Without replacements, the total cost would be brought down to $340.13.

3.3. Division of Manpower

This section talks about each team member's contributions towards the project. Each of the team members played a contribution in hardware, software, and business aspect of the project.

4. Testing

4.1. Conditions

In order to judge the capabilities of the module, several tests were performed under different conditions. Each condition represents a possible scenario that the Ball/Hole Locator Module may encounter during operation. These conditions include:

- A sunny day, with no clouds.
- A partly cloudy to cloudy day.
- Test times from midday to late afternoon where shadows are present.
- Looking at the ball from different directions with respect to the sun.
- At night using a flashlight for a light.
- Indoors: artificial lighting and a non-uniform background or Astroturf.

- Outdoors on varying types of grass.
- The module was mounted of 11" above the ground to simulate the chassis ride height.

These test conditions were used to characterize the performance of the module in a variety of different test conditions in addition to improving the overall performance of the module.

4.2. Tests performed

For each possible condition, it needed to be determined how far away could the ball be detected and tracked. Factors such as time of day and objects in the background that may cause the picture to be non ideal were noted. Also, both yellow and white golf balls were used to simulate different types of golf balls. The code was tested for stability by providing multiple objects in one picture and the object being tracked would often be removed to test module "smarts". Any thing that might cause code to get lock was considered.

The feedback for these tests came in the form of data. How far was the module able to track? Where did the code crash? Did the camera get distracted? This data was complied to give results on just when and where is the best place to use the module, and when is the module totally limited by its surroundings.

4.3. Results

After conducting the research and compiling data, the results have been decided. The ideal conditions for this module are a sunny afternoon with very few shadows and in short cut, uniform grass or Astroturf. Under these conditions, the camera could track accurately (continuous camera correction) of seven and a half feet. The program operated correctly when the object was taken away by restarting the scan. In the case of multiple objects, the camera would pick the bigger of the two objects.

For the more commonly occurring non-ideal conditions, the ability of the module varied greatly. During the late afternoon when shadows were long and also on cloudy days, the camera would sometimes end up tracking the grass instead of the ball. The reason for this was because the shadows would lower the average color value of the picture seen by the camera and thus grass now covered by the shadow would seem “bright”. Based on the searching algorithm, the camera would see “bright” as being the object to track. Also, range the range went down to be around five to six feet.

Tracking the ball indoors or at night was the hardest. Inside, the fluorescent lighting decreased the tracking range considerable. Also, there are many objects small enough inside that the camera would consider as a ball. Often, the program would get confused and track random objects in the room. The best solution was to place the camera on a uniform dark background or on the Astroturf. At night the flash light provided the light and the camera would not see the ball and instead track the light source. The program had no trouble tracking the flashlight since everything else was black.

The software did a good job of not tracking objects to0 big. Since the algorithm relies on size to track, large objects did not confuse the camera. Also, switching the white ball for a yellow one decreased the range farther since a yellow ball does not reflect the light as much. When the grass was not low cut or had holes the camera had a harder time finding the ball (often it would miss). Having the camera look at the horizon proved to not be a problem since the red channel was ignored in the search algorithm.

Based on the results of this data, one can conclude that the ball/hole locator module works best under outdoor conditions with few clouds. A background that is as uniform as possible also improves the range. All large objects will be ignored under these conditions as well. Under these outdoor conditions the module was determined to have a tracking range of five to seven feet. Thus, it is not highly recommend that the module be used in conditions where there are a lot of shadows or the background tends to be random (such as a lab) since can confuse the camera and the software. The color of ball can be taken into account by adjusting the confidence levels correctly in the program.

5. Conclusion

The final status of the Ball and Hole Locator is "Fully Functional". The capabilities include:

- Successfully locate a ball or hole at a distance of 2.25m
- Drive data is generated and sent to the chassis via Navigation
- The system is able to ignore visual distractions such as the sky or horizon. Objects similar to a ball or hole cannot be ignored.
- The system can dynamically track the ball or hole while the chassis is moving
- The system can run independently or integrated with the robot.

Conditions that affect the performance of the system include:

- Indoor lighting conditions (sunlight is beneficial)
- Colored balls cannot be easily tracked
- White objects within the field of view prevent tracking of target
- Long shadows that occur in the early morning and late afternoon may prevent tracking or limit range.

If this project were to be attempted again, it would be advisable allocate funds sooner as well as setting another deadline for individual functionality a few weeks prior to the final functionality deadline where full integration will be required.

Appendix A: Serial Communication Schematics and Altera Code

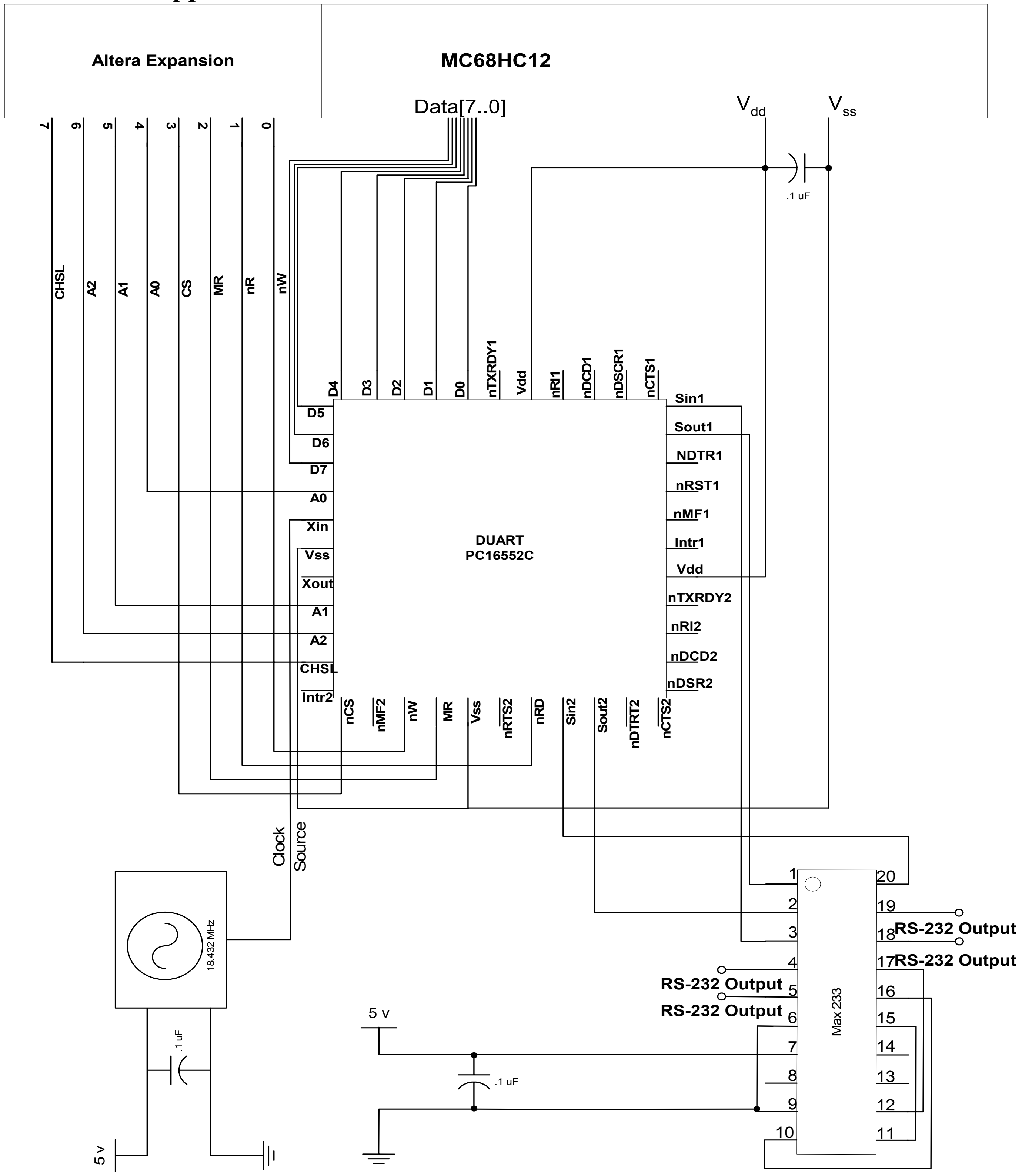


Figure 3: DUART Schematic. Note: Modem Pins should be wired high.

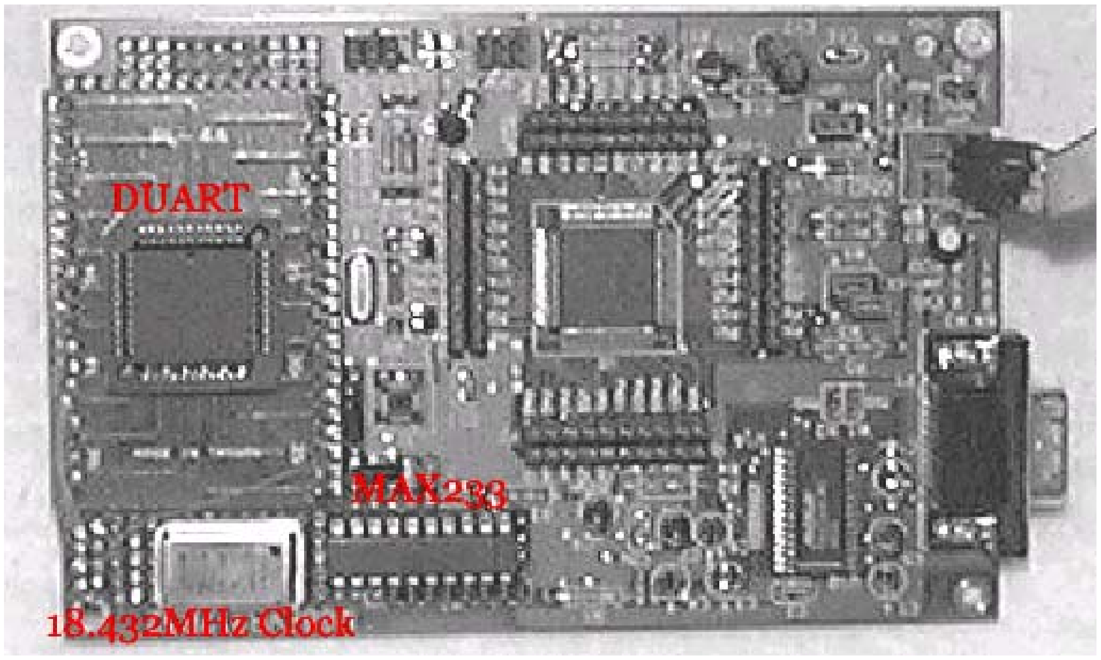


Figure 4: Component Layout for the DUART chip.

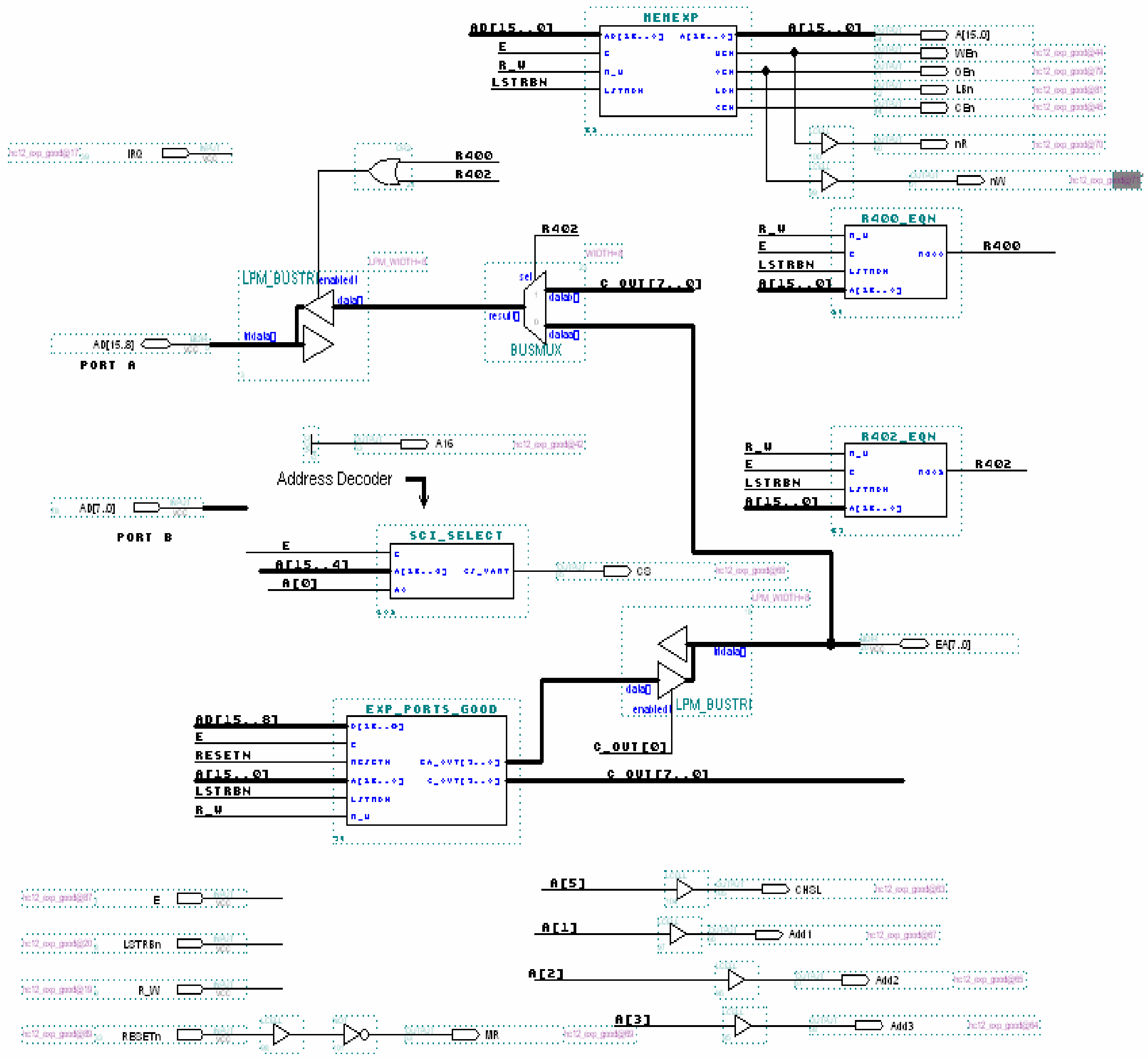
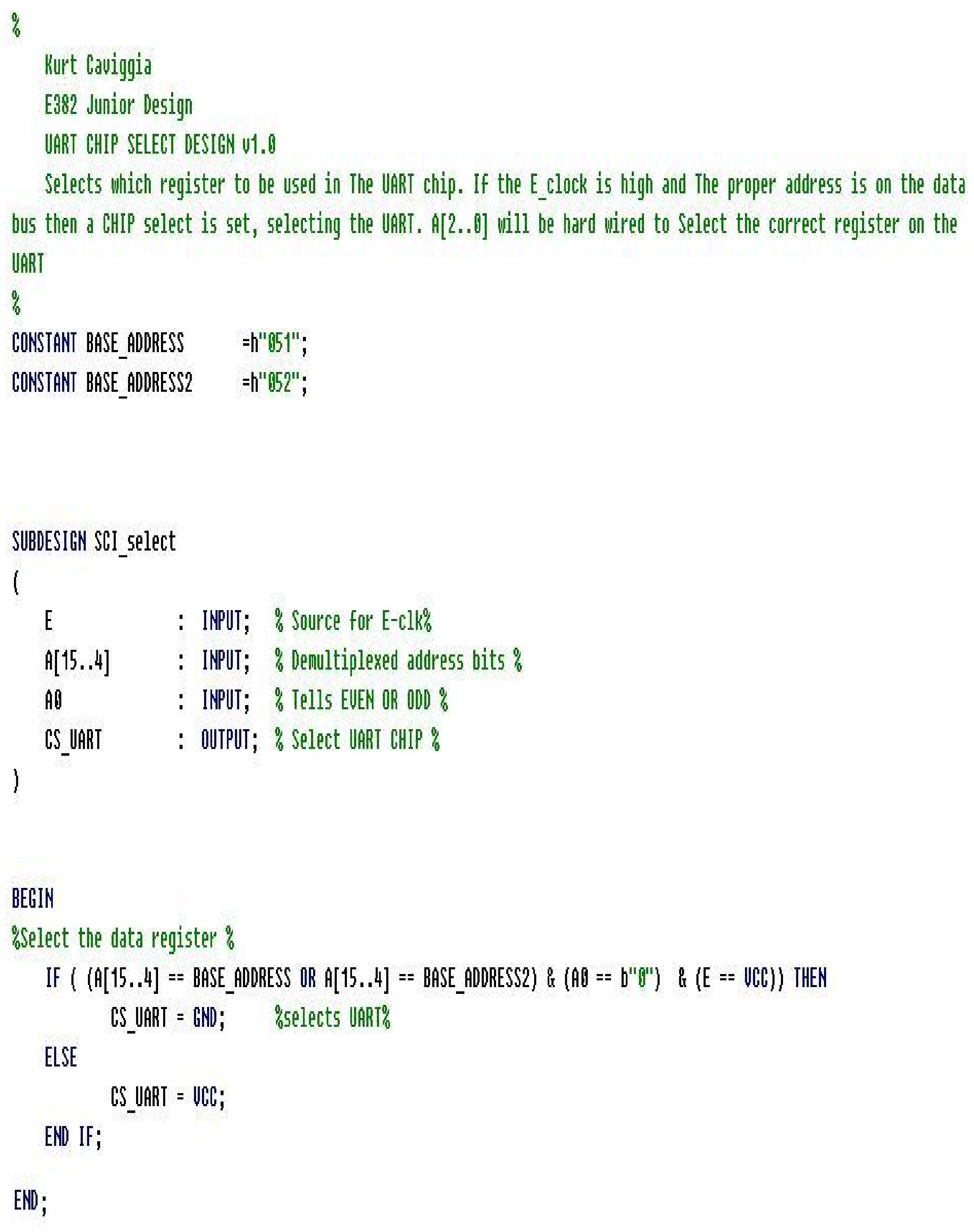
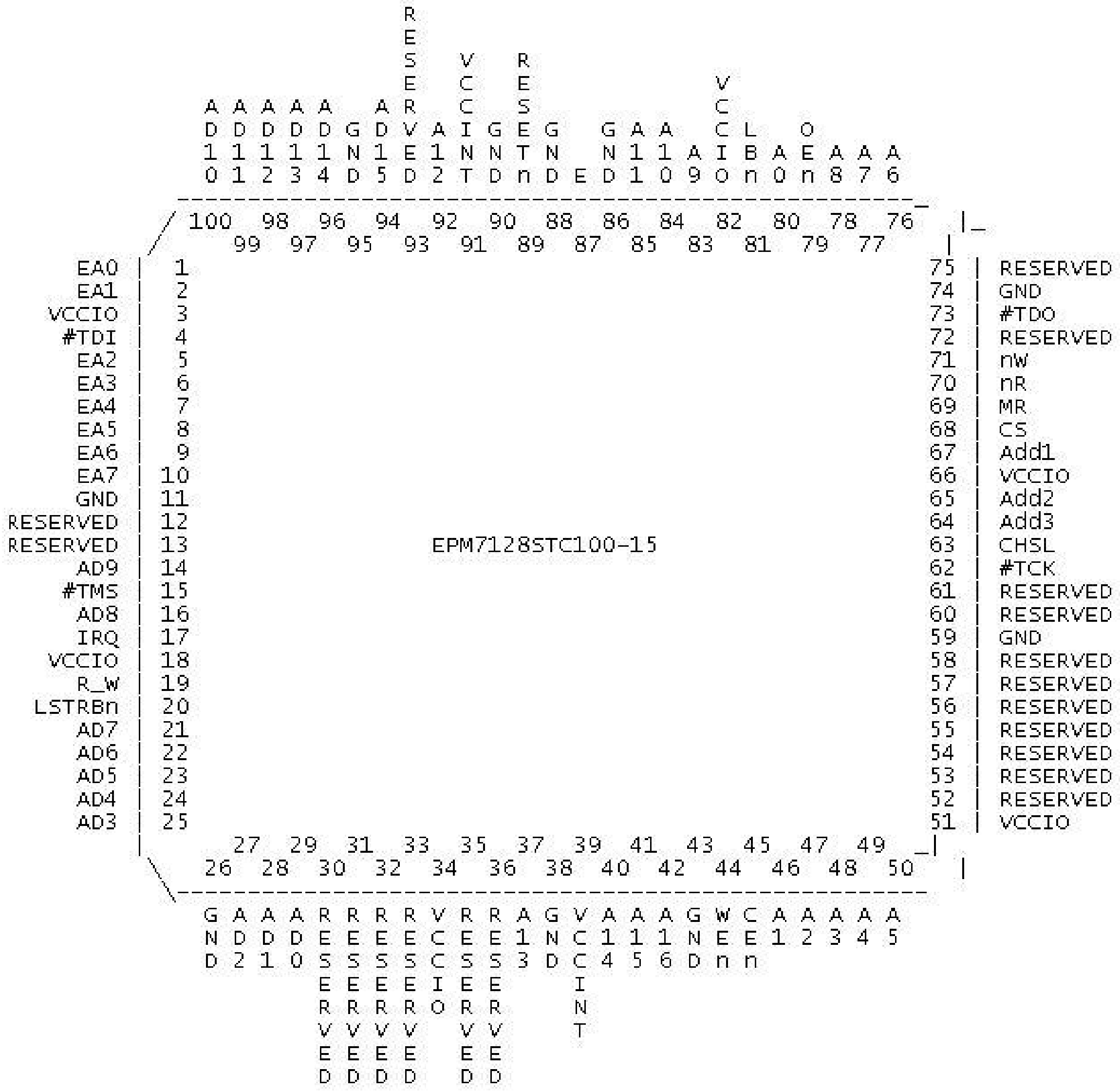


Figure 5: Final Altera gdf file including address decoder.



```
%
    Kurt Caviggia
    E382 Junior Design
    UART CHIP SELECT DESIGN v1.0
    Selects which register to be used in The UART chip. If the E_clock is high and The proper address is on the data
bus then a CHIP select is set, selecting the UART. A[2..0] will be hard wired to Select the correct register on the
UART
%
CONSTANT BASE_ADDRESS      =h"051";
CONSTANT BASE_ADDRESS2     =h"052";



SUBDESIGN SCI_select
(
    E             :  INPUT;   % Source for E-clk%
    A[15..4]      :  INPUT;   % Demultiplexed address bits %
    A0            :  INPUT;   % Tells EVEN OR ODD %
    CS_UART       :  OUTPUT;  % Select UART CHIP %
)


BEGIN
%Select the data register %
    IF ( (A[15..4] == BASE_ADDRESS OR A[15..4] == BASE_ADDRESS2) & (A0 == b"0")  & (E == VCC)) THEN
            CS_UART = GND;      %selects UART%
    ELSE
            CS_UART = VCC;
    END IF;

END;
```

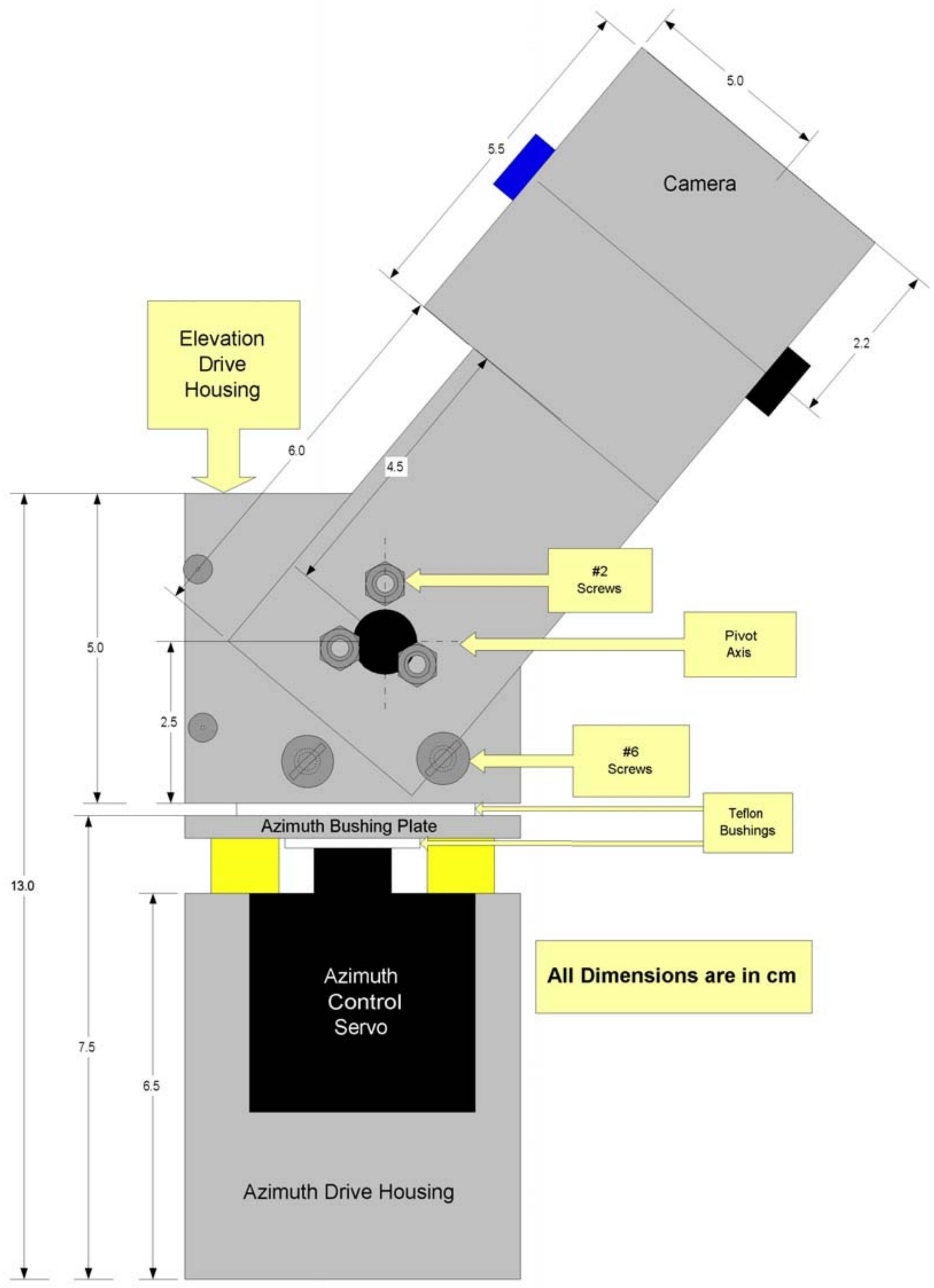
Figure 6: The address decoder .tdf file written in altera.



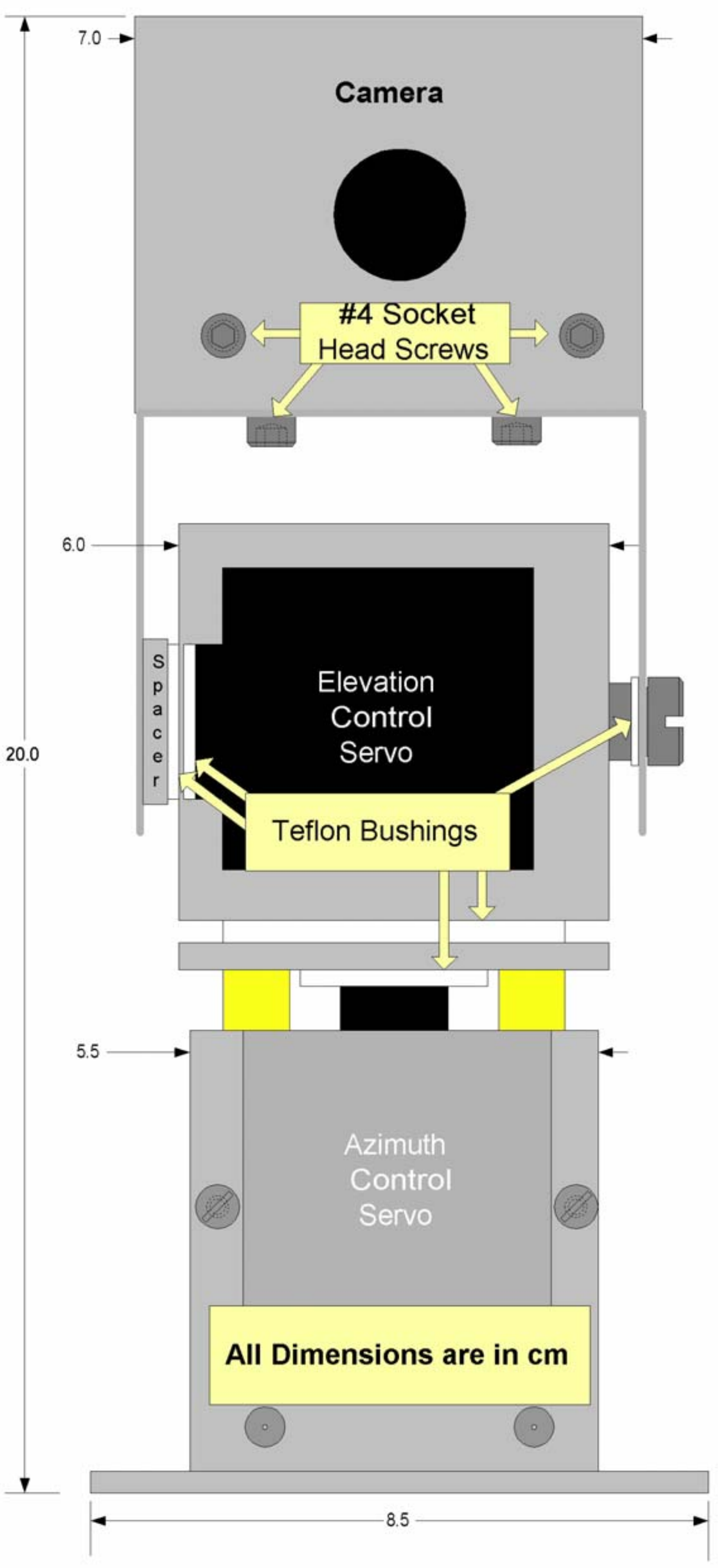
```
                                 R
                                 E
                                 S   V   R
                                 E   C   E             V
                   A A A A A   A R   C   S             C
                   D D D D D G D V A I G E G   G A A   C L   O
                   1 1 1 1 1 N 1 E 1 N N T N   N 1 1 A I B A E A A A
                   0 1 2 3 4 D 5 D 2 T D n D E D 1 0 9 o n 0 n 8 7 6
                 --------------------------------------------------_
              / 100  98  96  94  92  90  88  86  84  82  80  78  76¯  |_
             /      99  97  95  93  91  89  87  85  83  81  79  77     |
      EA0 |  1                                                      75 | RESERVED
      EA1 |  2                                                      74 | GND
   VCCIO  |  3                                                      73 | #TDO
     #TDI |  4                                                      72 | RESERVED
      EA2 |  5                                                      71 | nW
      EA3 |  6                                                      70 | nR
      EA4 |  7                                                      69 | MR
      EA5 |  8                                                      68 | CS
      EA6 |  9                                                      67 | Add1
      EA7 | 10                                                      66 | VCCIO
      GND | 11                                                      65 | Add2
 RESERVED | 12                                                      64 | Add3
 RESERVED | 13                   EPM7128STC100-15                   63 | CHSL
      AD9 | 14                                                      62 | #TCK
     #TMS | 15                                                      61 | RESERVED
      AD8 | 16                                                      60 | RESERVED
      IRQ | 17                                                      59 | GND
    VCCIO | 18                                                      58 | RESERVED
      R_W | 19                                                      57 | RESERVED
   LSTRBn | 20                                                      56 | RESERVED
      AD7 | 21                                                      55 | RESERVED
      AD6 | 22                                                      54 | RESERVED
      AD5 | 23                                                      53 | RESERVED
      AD4 | 24                                                      52 | RESERVED
      AD3 | 25                                                      51 | VCCIO
          |        27  29  31  33  35  37  39  41  43  45  47  49  _|
           \   26  28  30  32  34  36  38  40  42  44  46  48  50¯   |
            \--------------------------------------------------
                   G A A A R R R R V R R A G V A A A G W C A A A A A
                   N D D D E E E E C E E 1 N C 1 1 1 N E E 1 2 3 4 5
                   D 2 1 0 S S S S C S S 3 D C 4 5 6 D n n
                           E E E E I E E     I
                           R R R R O R R     N
                           V V V V   V V     T
                           E E E E   E E
                           D D D D   D D
```

Figure 7: Pin out for the programmed Altera expansion. EB is used for external pins.

Appendix B: Mechanical Drawings



Gimbal Side View



Gimbal Front View

Appendix C: HC12 Source Code

```
/************************  Main Program Last Update 4-26-02 @ 12:00pm ****************/
#include "hc12.h"
#include <stdio.h>
#include "DBug12.h"
#define YES 1
#define NO 0
#define TRUE 1
#define FALSE 0
#define GARBAGE 0x00

//CODES !!
//ff = done
//f0 = reposition
//0f = stop
//0x21 = find the ball/request acknowledge
/**************************SETUP AND DEFINE VARIABLES********************/

        /************* Prototyping Functions **********/
        void wait(void);                          /*indicator 5 */
        void resetcam(void);                      /*indicator 1 */
        void delay(int ms);                       /*ms is # of ms delay*/
        void verify(void);
        void scan(void);                          /*indicator 3 */
        void calcam(void);
        void getcolor(void);
        void trackball(void);
        void correctgimbal(void);                 /*indicator 2 */
        void acquireball(void);
        void acquirehole(void);
        void command(unsigned char send[]);       /*indicator 6 */
        void receive(int n);                      /*indicator 7 */
        void calcDRIVEdata(void);
        void calcSTEERdata(void);
        void send_data(unsigned char DATA);       /*indicator 4 */
        void receive_data(void);                  /*indicator 5 */

        /************ Prototype Interrupts**************/

        /************** Define Variables ***************/
        typedef enum {false,true} bool;
        volatile char RECEIVE = 0x00;

        signed int centerAZ = 40;  /*centroid of field of view (X)   */
        signed int centerEL = 71;  /*centroid of field of view (Y)   */

        unsigned int defaultAZ = 0x9d0;    /*home position of the gimbal     */
        unsigned int defaultEL = 0x666;    /*home position of the gimbal     */
        unsigned int final_AZB = 0x96e;    /*final position of gimbal        */
        unsigned int final_ELB = 0x43e;    /*final position of gimbal        */
        unsigned int final_AZH = 0x96e;    /*final position of gimbal        */
        unsigned int final_ELH = 0x43e;    /*final position of gimbal        */
        volatile signed int AZcor = 0;     /*error of azimuth(X)= 1-->80     */
        volatile signed int ELcor = 0;     /*error of elevation(Y)= 1-->143  */
        volatile signed int tempDTY0;       /*used for calculating DTY cor    */
        volatile signed int tempDTY2;       /*used for calculating DTY cor    */

        volatile unsigned int count;        /*placeholder for camdat array   */
        signed int camdat[15];              /*data going into and out of cam  */

        volatile char LOCK;                 /*LOCK=YES, conf>=50               */
        volatile char no_lock;              /*number of times LOCK = NO       */
        volatile char LOST;                 /*LOST=YES, Redo the scan         */
        volatile char TALK;                 /*TALK=YES, it is OK talk to NAV  */
        volatile char talk_delay =0; //used for testing and slowing down the data stream

        volatile char gotball = NO;/*keep  looking  for  ball*/
        volatile char gothole = NO;         /*keep  looking  for  ball*/
        unsigned char track_color[25];      /*used in track color command     */
        volatile char RED;                  /*notes the RED value for tracking*/
        volatile char GREEN;                /*notes the GRN value for tracking*/
        volatile char BLUE;                 /*notes the BLU value for tracking*/

        unsigned char DATA = 0;             /*represents data I/O from NAV    */

        volatile sending = FALSE;
        volatile finished = FALSE;
        char crap_received = 0;

        unsigned int DRIVEcor;              /*notes  size  of drive correction*/
        unsigned int STEERcor;              /*notes  size  of steer correction*/
        unsigned char FR;                   /*notes a fwd--rev      correction*/
        unsigned char LR;                   /*notes a left or right correction*/
        unsigned char STEER = NO;           /*decides to do a steer correction*/
        unsigned int z;
    /***********************************************************************/

void main(void)
{

    /**********************           PAD4 = Azimuth input  PAD5 = Elevation input*******/
        ATDCTL2 = 0x80;                     /* 1000 0000 */
        ATDCTL4 = 0x61;                     /* 011 00001 */
        ATDCTL5 = 0x34;                     /* x011 01xx */
    /***********************************************************************/

    /********************************** Use RTI *****************************/
        RTICTL = 0x87;                      //turn on RTI with 65ms
        RTIFLG = 0x80;                      //clear RTI flag
        INTCR = 0x00;
        disable();
    /***********************************************************************/

    /************************** Set up 2 PWM Channels***********************/
        PWCTL = PWCTL & ~0x80;     /*set to left-aligned */
        PWPOL = 0xF0;                 /*Select clock mode 1 CH: 0,2 */
        PWCLK = 0xC0;             /* N(1) and N(2) = 0*/
        PWSCAL0 = 1;             /*   M = 1 */
        PWSCAL1 = 1;
        PWPER0 = 40000;            /* Period 1 = PWPER1 + 1 */
```

```
		PWPER2 = 40000;		/* Period 0 = PWPER0 + 1 */
		PWDTY0 = defaultAZ;
		PWDTY2 = defaultEL;
		PWEN = 0x0F;			/*enables PWM CH 1 and CH 2*/

		/*PWDTY0  == Azimuth*/
		/*PWDTY2  == Elevation*/

	/*************************************************************************/

	/**************Setup for Input capture on TIC channel 5 ******************/
		TSCR = 0x80;	/*turn on timer*/
		TMSK2 = 0x03;	//65 ms
		TIOS = TIOS | 0x00; //gives us channel two as input capture
		TCTL4 = 0x10;	//SET UP T2 as rising
		TCTL3 = 0x04;	/*capture rising edge*/
		TMSK1 = 0x04;	// ENABLE INTERRUPTS CH2
		TFLG1 = 0x24;	/*Clear Flag on Pin 5*/
	/*************************************************************************/


	/************** SPI Communication to NAV ************************************/
		DDRS = DDRS | 0x10;	// ss, clk, MOSI outputs, MISO is input
		SP0CR1 = 0x4c;		// slave, MSB first, etc.
		SP0CR2 = 0x00;		// no baud rate register since master provides the clock

	/*************************************************************************/

	/************************** MISC SETUP ************************************/
		DDRT = (DDRT | 0xC0); /*make PortT[7] an output for camreset*/
		//THIS ALSO SETS UP PIN SIX AS OUPUT to MASTER!!!
		DDREA = 0xff;
		PORTEA = 0x00;
	/*************************************************************************/

	/* PORTT MAP=> T7:RESET CAM, T6:TO MSTR, T2&T3:INPUT CAPTURE  */

	/************************ SET UP SCI  **************************************/
	     LINE_CTR_ADDR = 0x80;		/* 1 0 0 0 0 0 1 1 */

    while(FIFO_ADDR != 0x00)
		{
			FIFO_ADDR = 0x00;	//Turns off the Concurrent write
		}

		LINE_CTR_ADDR = 0x80;

     while(RT_ADDR != 0x1e)
		{
	      LINE_CTR_ADDR = 0x80;
			RT_ADDR =  0x1e;	/* Sets The LSB of the divisor = 30(DLAB = 1)*/
		}


		IRQ_ADDR = 0x00;	/* Sets the MSB of the divisor = 0  (DLAB = 1)*/
```

```
                    /*Breaks the 18.432MHz down to a baud rate of 38400 bps */

                    DBug12FNP->printf("SCI Status: %x %x %x\n\r",RT_ADDR,IRQ_ADDR,
FIFO_ADDR);
                    LINE_CTR_ADDR = 0x03;
                    FIFO_ADDR = 0x01;

                    DBug12FNP->printf("FIFO: %s\r\n", ((FIFO_ADDR & 0xc0) == 0xc0) ? "on" : "off");

                    DBug12FNP->printf("SCI: %s\r\n", ((LINE_CTR_ADDR & 0x03) == 0x03) ? "ready" :
"failed!");

/****************************************************************************/

/************************************* MAIN PROGRAM ******************************/
          DBug12FNP->printf("Camera Tracker v.8.0 Waiting for request:>\n\r");

          //DATA = 0x21;
          //wait();            //Simulates Navigation in stand alone
while(1)                                  /*run the main programs forever*/
{
                    DATA = 0x00;
                    receive_data();
     /***** WAITS FOR REQUEST TO FIND BALL OR HOLE *****/

/******************** FIND THE BALL PROGRAM *********************/

          if(DATA == 0x21)                    /* run "find the ball" program */
          {
                    send_data(0x21); /*acknowledges request to find ball*/
                    DBug12FNP->printf("Ball Finder Started!! \n\r");
                    resetcam();
                    scan();

                    while(gotball == NO)
                    {
                    TALK = NO;
                              enable();
                              /*** This code sends out correction data to chassis ***/
                              /*** Large steering errors are fixed before driving ***/
                              /*** Then, steering and drive corrections alternate ***/
        if (talk_delay == 4)
          {
           TALK = YES;
           talk_delay = 0;
          }
                              if(TALK == YES)
                                        {
                                        if(PWDTY0  < 0x08C4 || PWDTY0 > 0x0A29)
                                                  {
                                                  STEERcor = final_AZB - PWDTY2;
                                                  calcSTEERdata();
                                           send_data(DATA);
                                                  DBug12FNP->printf("Ball in left field \n\r");
                                                  TALK = NO;
                                                  }
```

```
		}
	if(TALK == YES)
		{
		if(STEER == YES)
			{
			STEERcor = final_AZB - PWDTY2;
			calcSTEERdata();
			send_data(DATA);
			STEER = NO;
			DBug12FNP->printf("Steering.. \n\r");
			TALK = NO;
			}
		}
	if(TALK == YES)
		{
		if(STEER == NO)
			{
			DRIVEcor = ((PWDTY0 - final_ELB));
			calcDRIVEdata();
			send_data(DATA);
			STEER = YES;
			DBug12FNP->printf("Driving.. \n\r");
			TALK = NO;
			}
		}
	disable();
	delay(1);
	trackball();
	verify();

	if (LOCK == 1)
		{
		LOST = 0;
		no_lock = 0;
		correctgimbal();
		acquireball();
		}
	else
		{
		no_lock++;
		}
	if (VLIM > 0x08) /*Makes sure you're looking down*/
		{
		 scan();
		}
	if (HLIM > 0x0E || HLIM < 0x04) /*Makes sure you're not seeing chassis*/
		{
		 scan();
		}

	if (no_lock > 5)
		{
		send_data(0x0f);	/*Tells the robot to STOP!*/
		DBug12FNP->printf("STOP We lost the Ball \n\r");
		getcolor();
		LOST++;
```

```
                        }
                if (LOST > 15)
                        {
                         scan();
                        }
        }/*end of "gotball?" while */
        PORTEA = 0xA5;
        PORTEA = 0x00;

}/*end of "find the ball" Program */
/************************ END FIND THE BALL PROGRAM **********************/

/************************* FIND THE HOLE PROGRAM ************************/
if (DATA = 0x20)                        /*run "find the hole" program*/
{

 send_data(0x20);            /*acknowledges request to find ball*/
        DBug12FNP->printf("HOLE Finder Started!! \n\r");
        resetcam();
        scan();

        while(gothole == NO)
        {
          TALK = NO;
                enable();
                /*** This code sends out correction data to chassis ***/
                /*** Large steering errors are fixed before driving ***/
                /*** Then, steering and drive corrections alternate ***/
        if (talk_delay == 4)
                {
                TALK = YES;
                talk_delay = 0;
                }
                if(TALK == YES)
                        {
                        if(PWDTY0  < 0x08C4 || PWDTY0 > 0x0A29)
                                {
                                STEERcor = final_AZH - PWDTY2;
                                calcSTEERdata();
                                send_data(DATA);
                                DBug12FNP->printf("Hole in left field \n\r");
                                TALK = NO;
                                }
                        }
                if(TALK == YES)
                        {
                        if(STEER == YES)
                                {
                                STEERcor = final_AZH - PWDTY2;
                                calcSTEERdata();
                                send_data(DATA);
                                STEER = NO;
                                DBug12FNP->printf("Steering.. \n\r");
                                TALK = NO;
                                }
                        }
```

```
                    if(TALK == YES)
                            {
                            if(STEER == NO)
                                    {
                                    DRIVEcor = ((PWDTY0 - final_ELH));
                                    calcDRIVEdata();
                                    send_data(DATA);
                                    STEER = YES;
                                    DBug12FNP->printf("Driving.. \n\r");
                                    TALK = NO;
                                    }
                            }
                    disable();
                    delay(1);
                    trackball();
                    verify();

                    if (LOCK == 1)
                            {
                            LOST = 0;
                            no_lock = 0;
                            correctgimbal();
                            acquirehole();
                            }
                    else
                            {
                            no_lock++;
                            }
                    if (VLIM > 0x08) /*Makes sure you're looking down*/
                            {
                             scan();
                            }
                    if (HLIM > 0x0E || HLIM < 0x04) /*Makes sure you're not seeing chassis*/
                            {
                             scan();
                            }

                    if (no_lock > 5)
                            {
                            send_data(0x0f);        /*Tells the robot to STOP!*/
                            DBug12FNP->printf("STOP We lost the HOLE \n\r");
                            getcolor();
                            LOST++;
                            }
                    if (LOST > 15)
                            {
                             scan();
                            }
            }/*end of "gotball?" while */
            PORTEA = 0xA5;

   PORTEA = 0x00;

}/* end of "find the hole" Program*/
/*********************** END FIND THE HOLE PROGRAM **********************/
```

```
   DATA = 0x00;
  }/*end of while(1) loop */

}/*end of main*/
/********************************** END MAIN PROGRAM **************************/


/********************************** DEFINE Functions****************************/


        /****************************** WAIT Function ****************************/
        void wait()
        {
                PORTEA = 0x20;
                DBug12FNP->printf("waiting for trigger..\r\n");
                TFLG1 = 0x20; // clears flag for PT5
                while((TFLG1 & 0x20) == 0);        /*wait */
                PORTEA = 0x00;
        }
        /****************************END WAIT Function ***************************/



    /****************************RESETCAM Function****************************/
        void resetcam()
        {
                PORTEA = 0x01;
                PORTT = (PORTT | 0x80); //switch the relay TURN OFF CAM
                delay(1000);                    //wait
                TALK = NO;
                PORTT = (PORTT &~0x80);         //turn ON CAM
                delay(100);         // add some time
                command("RS\r");            //reset camera
                command("RM 3\r");          //turn on raw mode
                command("RM 3\r");
                command("PM 1\r");          //turn on poll mode
                command("PM 1\r");
                PORTEA = 0x00;
        }
        /*************************** END RESETCAM Function **********************/



        /****************************** DELAY Function **************************/
        void delay(int ms)
        {
                int i;
                while (ms>0)      /* Out loop delays num ms */

                {
                i = 1333;      /* Inner loop takes 6 cycles */
                while (i > 0)   /* 1333 times 6 = 1 ms */
                        {
                        i = i-1;
                        }
                        ms = ms - 1;
                }
}/*end of DELAY Function*/
/*************************************************************************/



/**************************** SCAN Function *******************************/
void scan()
{
            char sector;
            sector = 1;
            PORTEA = 0x04;
            LOST = 0;
            LOCK = 0;
            while(LOCK == 0)
              {


            if(sector == 1)
            {
             PWDTY2=0x63e;   /*cushion servo*/
            delay(250);
            PWDTY0=0x0CD1;          /*00000*/
            PWDTY2=0x06fb;          /*X0000*/
                        }
if(sector == 2)
      {
                        PWDTY0=0x0B28;          /*00000*/
                        PWDTY2=0x06fb;          /*0X000*/
                        }
if(sector == 3)
      {
                        PWDTY0=0x097F;          /*00000*/
                        PWDTY2=0x06fb;         /*00X00*/
                        }
if(sector == 4)
                        {
                        PWDTY0=0x07C5;          /*00000*/
                        PWDTY2=0x06fb;         /*000X0*/
                        }
 if(sector == 5)
            {
                        PWDTY0=0x05E9;          /*00000*/
                        PWDTY2=0x06fb;         /*0000X*/
            }
if(sector == 6)
            {
                        PWDTY0=0x05E9;          /*0000X*/
                        PWDTY2=0x0871;         /*00000*/
                        }
   if(sector == 7)
                        {
                        PWDTY0=0x07C5;          /*000X0*/
                        PWDTY2=0x0871;         /*00000*/
                        }
```

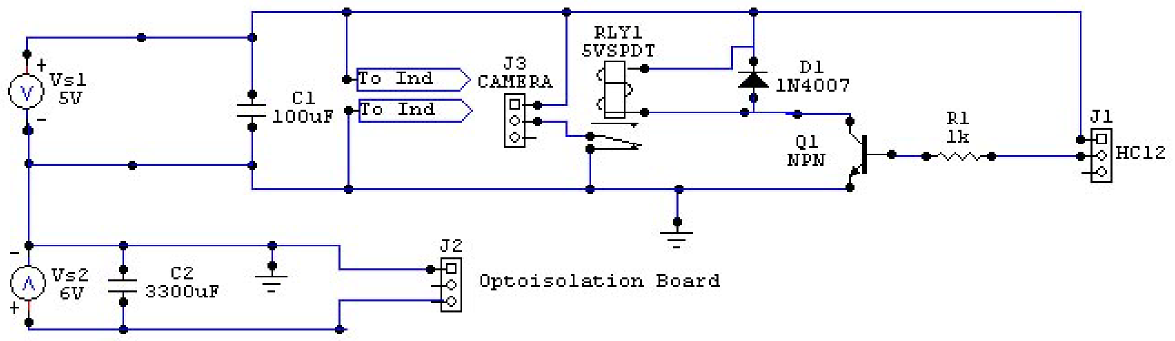
```
		if(sector == 8)
				{
				PWDTY0=0x097F;		/*00X00*/
				PWDTY2=0x0871;		 /*00000*/
				}
		    if(sector == 9)
				{
				PWDTY0=0x0B28;		 /*0X000*/
				PWDTY2=0x0871;		/*00000*/
				}
		 if(sector == 10)
				{
				PWDTY0=0x0CD1;		 /*X0000*/
				PWDTY2=0x0871;		 /*00000*/
				}

				if (sector == 10)   /*reset sector*/
					{
					//increment counter to slow down
					sector = 0;
					send_data(0xf0);  //tell robot to reposition
					DBug12FNP->printf("Standby while Robot repositions\n\r");
					delay (5000);
					}
				delay(500);
				calcam();
				getcolor();
				trackball();
				if(camdat[9] >= 50 && camdat[8] <= 50)
					{LOCK = 1;}
				sector = sector + 1;
				DBug12FNP->printf("Sector = %d Conf= %d Size= %d\n\r", sector, camdat[9],
camdat[8]);

		}/*end of "LOCK = NO" loop*/
		PORTEA = 0x00;
	}/*end of scan*/
	/***************************** END SCAN Function *************************/



	/*************************** CALCAM Function ***********************/
	void calcam()
	{
		command("CR 18 44\r");
		delay(2000);
		//command("CR 18 40\r");
	}
	/************************ END CALIBRATECAM Function **********************/


	/**************************** GETCOLOR Function **************************/
	void getcolor()
	{

		command("GM\r");
```

```
                receive(10);
                //RED   = camdat[3] + 2*camdat[6];
                GREEN = camdat[4] + 2*camdat[7];
                BLUE  = camdat[5] + 2*camdat[8];

        }
        /***************************************************************************/



        /**************************** TRACKBALL Function **************************/
        void trackball()
        {

                sprintf(track_color," TC 0 240 %d 240 %d 240\r",GREEN,BLUE);
                command(track_color);
                receive(10);
        //DBug12FNP->printf("Mx=%c My=%c Pix=%c Conf=%c
\n\r",camdat[2],camdat[3],camdat[8],camdat[9] );
        //DBug12FNP->printf("Mx=%d My=%d Pix=%d Conf=%d
\n\r",camdat[3],camdat[4],camdat[8],camdat[9] );
        //DBug12FNP->printf("1=%c 2=%c 3=%d 4=%d 5=%d 6=%d 7=%d 8=%d 9=%d 10=%d\n\r",

        //camdat[1],camdat[2],camdat[3],camdat[4],camdat[5],camdat[6],camdat[7],camdat[8],camdat[9],
camdat[10] );
                //DBug12FNP->printf("ELEVATION =%x \r\n", PWDTY2);

        }
        /************************** END TRACKBALL Function ************************/



        /*************************** ACQUIRE BALL Function *************************/
        void acquireball()
                {

        if(((PWDTY0 | 0xfff0)==(0xfff0 |final_AZB)) && ((PWDTY2 | 0xfff0)==(0xfff0 | final_ELB)))
                {
          gotball = YES;
                 PORTEA = 0xFF;
                 DBug12FNP->printf("We have a ball !!! \n\r");
             send_data(0xff);
                 DATA = 0x00;
                 PORTEA = 0x00;
                }
                else
                {gotball = NO;}

                }
        /************************* END ACQUIRE BALL Function **********************/

        /*************************** ACQUIRE HOLE Function *************************/
        void acquirehole()
                {

        if(((PWDTY0 | 0xfff0)==(0xfff0 |final_AZH)) && ((PWDTY2 | 0xfff0)==(0xfff0 | final_ELH)))
                {
                 gothole = YES;
                 PORTEA = 0xFF;
                 DBug12FNP->printf("We have a HOLE!!! \n\r");
             send_data(0xff);

                 PORTEA = 0x00;

                }
                else
                {gothole = NO;}

                }
        /************************ END ACQUIRE HOLE Function **********************/

        /*********************************** CORRECT GIMBAL Function *************/
        void correctgimbal()
                {
                PORTEA = 0x02;
                /**********Calculate El and Az Correction ***********/
                AZcor = camdat[3] - centerAZ;
                ELcor = camdat[4] - centerEL;
                //DBug12FNP->printf("AZcor = %d  ELcor = %d \n\r",AZcor,ELcor);
                /******** Scale and Write Duty cycle Correction  ****/
                tempDTY0 = PWDTY0 + AZcor*7;
                tempDTY2 = PWDTY2 - ELcor*3;
                PWDTY0 = (unsigned int)(tempDTY0);
                PWDTY2 = (unsigned int)(tempDTY2);

                PORTEA = 0x00;
                }/*end of "correct gimbal" routine */

        /************************** END CORRECT GIMBAL Function *****************/



        /***************************Begin DRIVE Function************************/
        void calcDRIVEdata()
                {
                FR=0;
                if ((DRIVEcor & 0x8000))
                        {
                        FR = 1;             /* Go in Reverse */
                        DRIVEcor = DRIVEcor - 32768;
                        }
                DATA = (DRIVEcor >> 5);
                if (FR == 1)
                        {
                        DATA = DATA | 0x20;    /*Tells chassis to reverse */
                        }
                DATA = DATA | 0x80;            /*Tells chassis, to go */
                }
        /**********************End DRIVE Function*********************************/
```

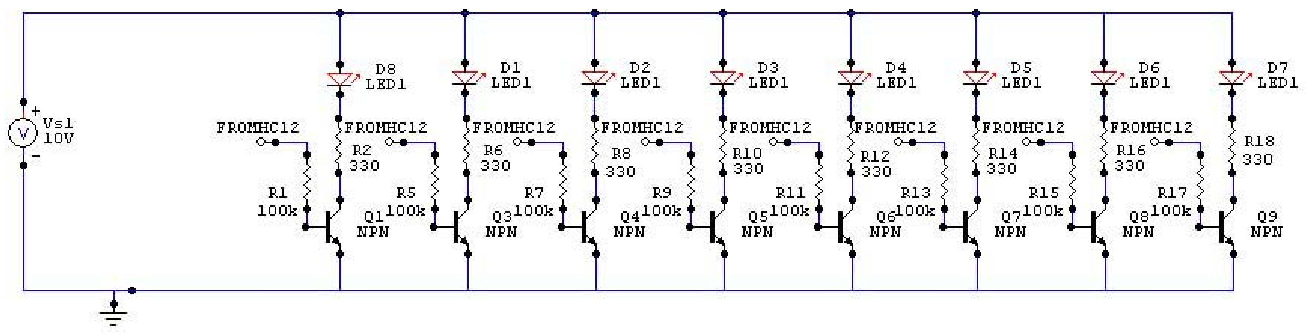
```
    /**********************Begin calcSTEERdata Function ***********************/

    void calcSTEERdata()
            {
        LR=0;
            if ((STEERcor & 0x8000))
                    {
                    LR = 1;             /* Turn Left */
                    STEERcor = STEERcor - 32768;
                    }
            DATA = (STEERcor >> 5);
            if (LR == 1)
                    {
                    DATA = DATA | 0x20;     /*Tells chassis to turn left */
                    }
            DATA = DATA | 0x40;             /*Tells chassis to turn */
            }
    /**********************END of STEER Function****************************/

  /*********************** RECEIVE DATA FROM NAV *************************/

    void receive_data(void)
    {

            PORTEA = 0x20;
            DBug12FNP->printf("waiting for nav \n\r");

            RTICTL = 0x00;                  //turn OFF RTI with 65ms
            TMSK1 = 0x04;                   // ENABLE INTERRUPTS CH2
            enable();
            //while(!RECEIVE);              //WAIT until TIC2 HANDSHAKE SET

            PORTT = PORTT | 0x40;                   // raise handshake
            SP0DR = GARBAGE;                        // send back garbage

            while ((SP0SR & 0x80) == 0);        // wait for transfer to finish

            DATA = SP0DR;                       // read out the received data, and this also
clears

            PORTT = PORTT & ~0x40;                          // drop handshake

            RECEIVE = FALSE;                // done receiving
            TMSK1 = 0x00;                   // DISABLE INTERRUPTS CH2
            RTICTL = 0x87;                  //turn on RTI with 65ms
            PORTEA = 0x00;
            disable();
            DBug12FNP->printf("DATA input from NAV = %x\n\r", DATA);
    }
  /*********************** END RECEIVE FROM NAV ***************************/


  /*********************** Send DATA to NAV *******************************/

    void send_data(unsigned char DATA)
    {
            PORTEA = 0x10;
            DBug12FNP->printf("Sending %u \n\r", DATA);

             SP0DR = DATA;

            PORTT = PORTT | 0x40;  // raise line to tell master we want to transfer

            while ((SP0SR & 0x80) == 0);            // wait for transfer
            crap_received = SP0DR;                  // clear spi flag
            PORTT = PORTT & ~0x40;                  // drop handshake line
            DBug12FNP->printf("DATA output = %x\r\n", DATA);
            PORTEA = 0x00;

    }
    /************************ END Send to NAV ***************************/


    /************************** RECEIVE Function *****************************/
    void receive(int n)
    {

            PORTEA = 0x80;
            count = 0;
            while(count != n)
                    {
                    while((IRQ_ADDR & 0x04) != 0x04);
                camdat[count] = RT_ADDR;
                    count++;
                    }
            PORTEA = 0x00;
    }
    /************************ END RECEIVE Function ***************************/


    /*************************** COMMAND Function ***************************/
    void command(unsigned char send[])
    {
            PORTEA = 0x40;
            count = 0;
            while(send[count] != '\0')
                    {
                    RT_ADDR = send[count];
                    delay(5);
                    count++;
                    }

            PORTEA = 0x00;
    }
    /************************ END COMMAND Function *************************/


    /*************************** VERIFY Function ************************/
    void verify()
    {
```

```
    if(camdat[9] >= 50 && camdat[8] <= 50)
            {LOCK = YES;}
    }

/************************* END VERIFY Function ***********************/



/******************************* END DEFINE Functions ***************************/


/****************************   DEFINE   Interrupts *****************************/
@interrupt void rti_isr(void)
        {
        PORTEA = 0x08;
        talk_delay++;
        RTIFLG = 0x80;                  //clear RTI flag
        PORTEA = 0x00;
        }
@interrupt void tic2_isr(void)
        {
        RECEIVE = TRUE;
        DBug12FNP->printf("TIC 2 interrupt \n\r");
        TFLG1 = 0x04;                   // clear the channel 2 flag
        }
/***************************  END DEFINE Interrupts ***********************/
```

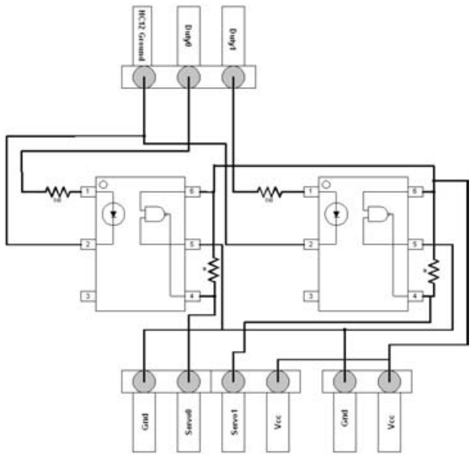
Appendix D: Schematics



Power Distribution Board.

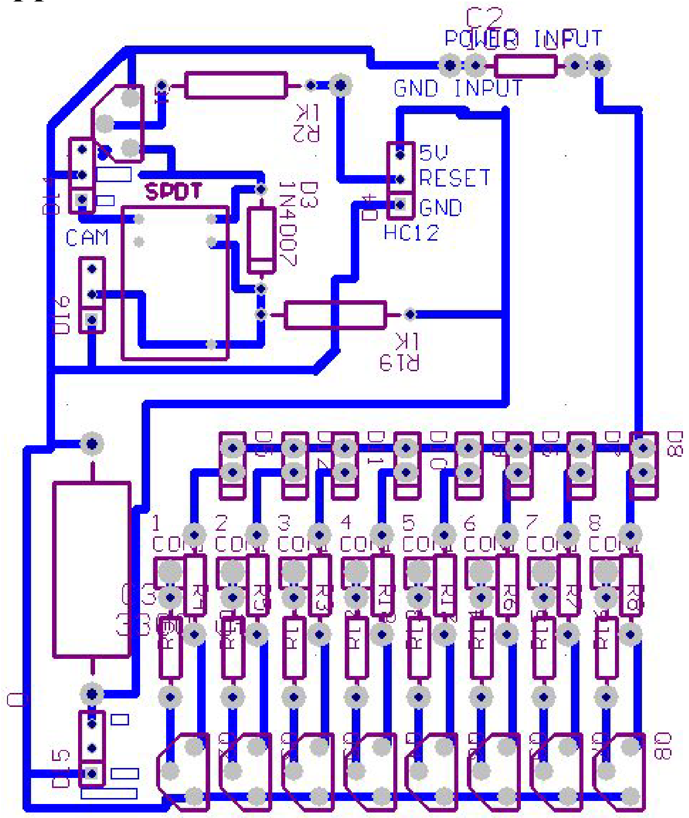


Indicator Pannel on the Power Distribution Board.

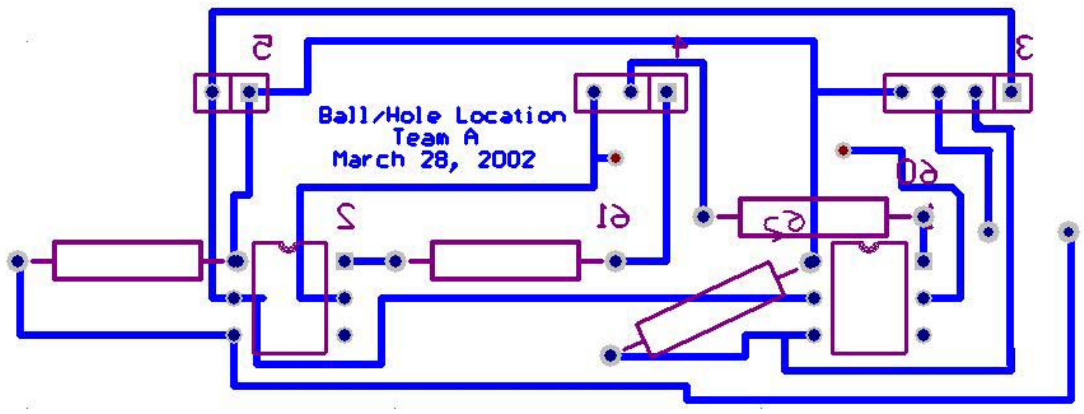


Optical Isolator Schematic

Appendix F: Printed Circuit Board Artwork

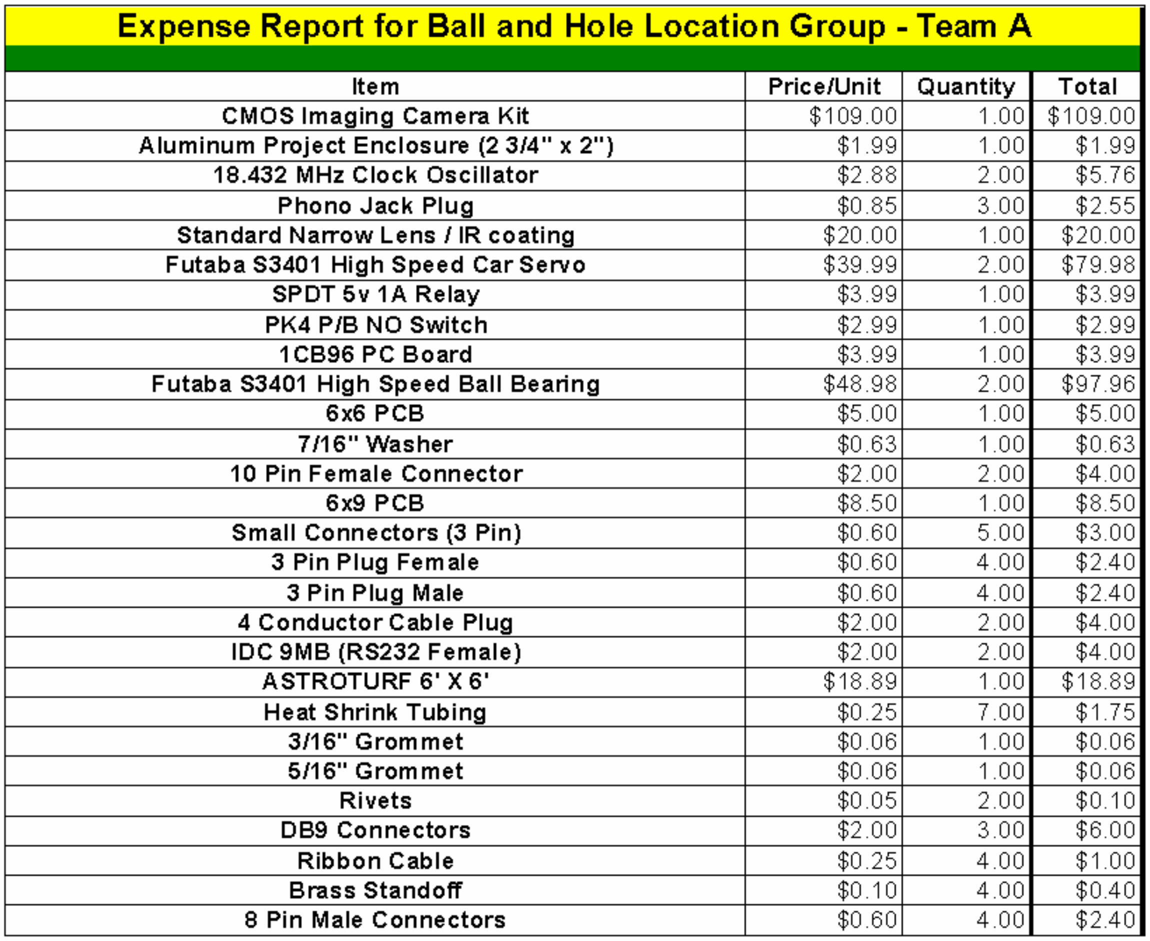


Power Distribution Board Layout



Optical Isolation Board Layout

Appendix F: Project Costs



Expense Report for Ball and Hole Location Group - Team A

Item	Price/Unit	Quantity	Total
CMOS Imaging Camera Kit	$109.00	1.00	$109.00
Aluminum Project Enclosure (2 3/4" x 2")	$1.99	1.00	$1.99
18.432 MHz Clock Oscillator	$2.88	2.00	$5.76
Phono Jack Plug	$0.85	3.00	$2.55
Standard Narrow Lens / IR coating	$20.00	1.00	$20.00
Futaba S3401 High Speed Car Servo	$39.99	2.00	$79.98
SPDT 5v 1A Relay	$3.99	1.00	$3.99
PK4 P/B NO Switch	$2.99	1.00	$2.99
1CB96 PC Board	$3.99	1.00	$3.99
Futaba S3401 High Speed Ball Bearing	$48.98	2.00	$97.96
6x6 PCB	$5.00	1.00	$5.00
7/16" Washer	$0.63	1.00	$0.63
10 Pin Female Connector	$2.00	2.00	$4.00
6x9 PCB	$8.50	1.00	$8.50
Small Connectors (3 Pin)	$0.60	5.00	$3.00
3 Pin Plug Female	$0.60	4.00	$2.40
3 Pin Plug Male	$0.60	4.00	$2.40
4 Conductor Cable Plug	$2.00	2.00	$4.00
IDC 9MB (RS232 Female)	$2.00	2.00	$4.00
ASTROTURF 6' X 6'	$18.89	1.00	$18.89
Heat Shrink Tubing	$0.25	7.00	$1.75
3/16" Grommet	$0.06	1.00	$0.06
5/16" Grommet	$0.06	1.00	$0.06
Rivets	$0.05	2.00	$0.10
DB9 Connectors	$2.00	3.00	$6.00
Ribbon Cable	$0.25	4.00	$1.00
Brass Standoff	$0.10	4.00	$0.40
8 Pin Male Connectors	$0.60	4.00	$2.40

LIST OF FREE PARTS

Item	Quantity
100k Ohm Resistors - 1/4 Watt	8.00
10k Ohm Resistors - 1/4 Watt	2.00
1k Ohm Resistors - 1/4 Watt	3.00
510 Ohm Resistors - 1/4 Watt	2.00
330 Ohm Resistors - 1/4 Watt	8.00
NPN Darlington Transistors	9.00
3300uF Capacitor	1.00
100uF Capacitor	1.00
470uF Capacitor	2.00
20mA LED	8.00
H11L1 Optoisolators	2.00
3"x5"x7" Enclosure	1.00
RJ45 Female Connector	1.00
6"x6" Perforation Board	1.00
4 Pin DIP Switch	1.00
1N4007 Diode	1.00
HC12 Microcontroller Board	1.00
MAXIM 233 IC	1.00
16552 DUART	1.00
Solder Mount Pin Headers	2.00
6 Pin Solder Lead Sockets	2.00
2"x3" Ribbon Cable Connector	2.00
12" x 12" Sheet 18 Gauge Aluminum	3.00
4/40" Screws	10.00
Brass Spacers	10.00
1/4" Bolt and Nut	1.00
7/8" Washers	4.00
30 mil Teflon Sheet	1.00
On/Off Switch	1.00