Vision Controlled NIOS-Bot
ViCoN-Bot

Jeff Vickers
gte613i
Kay Bosompem
gte141h
Kevin Walker
gte143x
Andre Moore
gt6875a

ECE 4006: Senior Design
Professor: James Hamblen
Summer 2002

July 25, 2002

Georgia Institute of Technology
School of Electrical & Computer Engineering
# Table of Contents

EXECUTIVE SUMMARY ................................................................. - 1 -
INTRODUCTION ........................................................................ - 2 -
OVERVIEW ................................................................................ - 2 -
DIGITAL CAMERA ................................................................. - 3 -
NIOS & CMUCAM INTERFACE .................................................. - 8 -
PULSE MODULATION WITH THE NIOS BOARD ......................... - 16 -
ROBOT DESIGN ........................................................................ - 18 -
STATE MACHINE ....................................................................... - 20 -
TESTING .................................................................................... - 23 -
ORGANIZATION ......................................................................... - 26 -
CONCLUSION ............................................................................. - 27 -
APPENDIX A: ............................................................................ A - 29
APPENDIX B: ............................................................................. B - 33
APPENDIX C: ............................................................................. C - 54
APPENDIX D: ............................................................................. D - 69
APPENDIX E: ............................................................................. E - 80
The ViCoN-Bot is a name coined for a visually controlled, Nios operated, robot, and was realized using a digital camera, an Altera Nios FPGA board, and a 1/12 scale electric Hummer. The mission of the ViCoN-Bot was to traverse a designed course using a digital camera as of means of detection for a series of unique and defining landmarks. Ideally, the EE hallway would serve as a full scale course, and future robots would compete in a race based on a design pioneered by the ViCoN-Bot.

Design of the ViCoN-Bot consisted of research and planning, software development, hardware implementation, and testing. Extensive research of the camera and its functionality was conducted to acquire a prior knowledge necessary to develop software controlling the camera. The software development was comprised primarily of a camera interface, robot speed and direction control, and a state machine. Hardware execution involved mounting and properly interconnecting all devices to the robot and providing sufficient power sources for each module. Integration of the previous phases led to the testing phase. The track consisted of neon pink landmarks symbolizing forward waypoints, and neon green landmarks denoting when to begin a full right hand turn. The ViCoN-Bot demonstrated the ability to find and follow the neon pink markers as well as properly negotiate a turn upon finding neon green. Future projects could easily build upon the framework set forth with the original design of the ViCoN-Bot. Adding secondary peripheral devices, improving existing hardware modules, and implementing more landmarks are among some of the improvements that could be introduced in future projects.
Introduction

The ViCoN-Bot is a standard RC car modified for guidance with a digital camera and controlled using an FPGA board. Design of the ViCoN-Bot consisted of mastering the control of a digital camera, programming an interface between the camera and the FPGA, coding a FPGA module to control the robot, constructing a robot, developing a state machine, assembling a course, and complete testing and debugging. The goal of the ViCoN-Bot was simply to find and drive to neon pink landmarks and turn at green landmarks. Accomplishing this would allow for a unidirectional course to be devised using a series of these landmarks as waypoints. By meeting this goal, the ViCoN-Bot is certain to lay the foundation for more advanced senior designs that utilize digital cameras in analogous ways.

Overview

The design of the ViCoN-Bot began with researching and selecting the proper components and equipment. Due to readily available supply and support locally, the Tamiya M1025 Hummer and Altera Nios board were the robot and FPGA board of choice. Proceeding significant research, the CMUcam was elected the most logical choice of a digital camera offering basic functionality integrated into the unit itself. Complete documentation provided in the CMUcam manual aided in developing a comprehensive understanding of the camera. Three battery sources were chosen to power the Hummer, Nios board, and camera. An additional voltage regulator circuit was implemented to supply the robot with a reference potential. A full parts list of all basic components is shown below in Table E-1.

The basic design consisted of several units all utilizing the primary processor. The Nios board provided the logic for both the system processor and additional units designed in C and VHDL. The logic blocks implemented on the Nios board consisted of a basic driver or state machine coordinated with the processor, a UART or camera interface, a speed control, and a direction control unit. The camera interface received data packets from the camera mounted on top of the robot, output the data to the state machine, and transmitted commands to the camera from the state machine. The state machine used the data packets to determine the robot’s orientation with its surrounding environment and compute the proper reaction of the robot. The computation was recorded in the form of two integer values ranging from 0 to 8, one for speed and one for direction. Both integers were output to corresponding pulse width modulation,
PWM, control units in the form of a 3-bit number corresponding to eight speed and eight directions settings. Each control unit deciphered the 3-bit input as a pulse length or duty cycle and generated a 20 ms output pulse of equivalent proportion. The PWM signals were routed to the robot’s speed controller which directly controls the robot. A diagram outlining the basic operation and flow of the ViCoN-Bot is shown below in Figure 1.

![Diagram of the ViCoN-Bot's overall operation and data flow.](image)

Once all functional units were assembled and jointly operational, a track consisting of chosen objects was developed. Through research of the CMUcam, neon pink and green rectangular objects were determined to be landmarks for a unidirectional course. Guided by the CMUcam the Hummer could traverse the course by seeking landmarks and reacting to them as the state machine was programmed. This arrangement collectively led to the construction of the ViCoN-Bot.

**Digital Camera**

After much research, the camera module of choice was a low-cost, low-power CCD camera developed by Carnegie Mellon. This unit was developed for robotics purposes and was
code named CMUcam as pictured in Figure 2. The camera itself is an Omnivision OV6620, single-chip CMOS CIF color digital camera. Coupled with an 80 MHz Scenix Programmable Integrated Circuit or PIC, the unit was capable of preprocessing the image and performing minor calculations. Processing the raw images instantaneously limited the overall resolution to 80 pixels wide by 143 pixels high and a maximum of 17 frames per second. The CMUcam communicated on a serial bus and ASCII interface with a computer via a standard telnet application or a Java GUI packaged with firmware v1.12 of the software for CMUcam. A null-modem was needed to communicate with the Nios board properly. A ¼” wide angle IR lens was added to filter out infrared and noisy light emitted from the sun and incandescent lights and also widened the field of view to 25°. The CMUcam required approximately 6 volts DC and 200 milliamps. Full CMUcam specifications including complete parts list, IC layout, schematic, full commands, packet formats, and user manual are referenced in the ViCoN-Bot CD.

The CMUcam required excessive experimentation and testing to properly master its functionality and understand its reaction to different environments. The first step was finding the focus point of the IR lens. It was determined by dumping frames that the sharpest focus of the camera was achieved at precisely 2 ¾ revolutions from completely screwed in. Once sharp images could be taken, the next challenge was determining the effects that different lighting conditions had on the image colors. Even with the aid of the IR lens, it was found that the sun and incandescent lights still had a large effect on the colors of the image; however, without the filter, images were very bright, blurry, and almost unrecognizable. Figure 3 shows the difference between the pictures taken the length of the EE hallway and from the same location facing the windows.
Figure 3. Pictures taken from the same location facing the sun, left, and an incandescently lighted hallway, right.

The most uniform color patterns were obtained in fluorescent light and areas of indirect light. The CMUcam has some basic commands that allow tweaking of the color settings to an extent. This is especially important when in tracking mode. Not only were there differences in colors of different lighting conditions, but also with varying the distance from which the image was captured. All of this was taken into consideration when choosing color ranges for tracking objects. At first thought, black and white objects were chosen as test color ranges. Black and white objects did not provide a good contrast as the camera computes colors based on an average. Grey is the naturally the average color in most images and the factor that limited the contrasting ability of black and white. Listed in order from best to worst, it was determined by experimentation that neon yellow, neon pink, and neon green had the most defined color ranges and contrasted well with surrounding colors. However, neon yellow was immediately eliminated as an option due to the fact that other light sources, especially incandescent, drastically interfered causing the object boundaries to become less defined and often random. At distances varying from three to twelve meters, neon pink, green, and yellow colored posters were recorded and viable track ranges were determined from the minimum and maximum values recorded from each color. The experimental data used to compute the color ranges are stored in Table A-1 through Table A-3. It is notable that neon yellow was not able to produce a traceable image at twelve meters due to noise from the overhead incandescent lights in the EE hallway. From this data, it was determined that the best trackable color, neon pink, be used as the color for guidance landmarks and neon green used as the color for landmarks designating turns.

Once color ranges were determine, the objects used as landmarks were determined including size, shape, and orientation. Determining the proper landmarks parameters required a complete knowledge of how the CMUcam tracks objects. Tracking was done with a simple
command and the color range as an input. The CMUcam then searches for the input color range and calculates the square area that fits that color range. Once the object is located, several calculations are returned including the coordinates of the two corners defining the object, the area they encompass, the middle mass of the object, and the confidence in which the object fits the input color range. The provided GUI shows the object it is tracking with a green, yellow, or red rectangle and the middle mass with a little red square. The color of the rectangle is determined by how confident the object is tracking, green being the best. The coordinates are useful in determining the orientation of the object in the camera’s field of view. The area is functional in calculating the distance of a known object from the camera. The middle mass serves as the guiding coordinate for tracking. And the confidence determines how well the object is tracking as well as means for calculating the speed of the robot. Figure A-1 shows a picture of two posters before tracking, top, while tracking green, center, and while tracking pink, bottom, from three to fifteen meters. This figure shows that the neon green poster could be confidently tracked out to twelve meters and vaguely tracked at fifteen meters. Similarly, the neon pink poster was tracked confidently at distances exceeding fifteen meters. The ability of neon pink to be tracked more confidently at greater distances reconfirmed our previous inclination to use neon pink for the most critical and reoccurring landmark. A half poster of neon pink was determined to be extremely trackable at distance exceeding ten meters; therefore, ten meters was chosen to be the distance separating guiding landmarks. A full neon green poster is used for a land marker denoting a turn.

Once the robot is traversing the course, turning is the next item of concern. Turning requires a more calculations than mere alignment as with the neon pink guidance markers. The distance from the turning landmark is crucial in determining when to slow down and begin turning the robot. Size is the most useful piece of information in calculating distance. Since landmarks are of known size, color, and location, they can be located and the range determined by data gathered from the camera. If an object is of known size in an image, the distance could be calculated precisely; however, the relatively low resolution of the image and high speed of the robot make this exact calculation less relevant. Rather, it is more useful to predetermine the size of the image at which a deceleration and turn is appropriate. Size and confidence measurements were taken from 1 meter to 5.5 meters in ½ meter increments and the data stored in Table A-1.

One major problem seen while recording this data relates to the earlier problem of varying colors
at different distances. Unexpectedly, the size and confidence measurements became increasingly erratic at distances less than 2.5 meters. **Figure 4** demonstrates this by showing the same object being realized as three different sizes and confidences in different frames while tracking. Contrary to reason, the CMUcam took more accurate measurements from distances greater than 2.5 meters. In fact, the size was nearly modeled as a linear function of distance in between 2.5 and 5.5 meters. Therefore, the sizes used in negotiating turns will be calculated at distances that remain in between 2.5 and 5.5 meters.

**Figure 4.** Tracking errors common associated with close proximities.

Not only does **Figure 4** demonstrate an error associated with a combination of close proximity measurements, interference from an incandescent light source, interference from the sun, and noise from the green tree in the background, but it also shows the CMUcam’s innate ability to decipher the true location of the object. Despite several types of inference and sources of error that could possibly exist in this situation, the middle mass locations are always in precisely the actual location. This feature demonstrates the rigidity of the CMUcam. Overall, the CMUcam proved ideal for robotics as the ViCoN-Bot verified.
Nios & CMUcam Interface

After deciding on what camera and robot to use for the ViCoN project, the computing power required became the final piece of our hardware. This segment is responsible for making the robot autonomous. A small, lightweight, yet powerful computer with an assortment of standard peripherals was required. A Field Programmable Gate Array (FPGA) or a microcontroller would be fairly adequate. Initial research narrowed the list of possibilities for the ViCoN-Bot to the following:

- Microchip PICF16787
- Microchip PIC18F452
- Microchip PIC18F442
- Xilinx board
- Nios® CPU development board

Based on accessibility, memory size, processor speed and available of peripherals, the Nios® CPU development board was the most appropriate choice. The Nios® CPU development board will hitherto be referred to as the Nios board. The Altera Nios board provides all the necessary tools to create a highly configurable embedded system. This board consists of three basic components: programmable logic, memory and a processor core. The processor core, after which the entire system is named, is called Nios. Nios is a customizable soft processor core. This runs atop an ARM RISC based processor.

![Figure 5. Arm based Excalibur Embedded Processor.](image)
Figure 5, shown above, depicts a high level representation of the embedded processor core. All components in the interior of the core processor can be configured to required specifications.

Figure 6, shown below, illustrates a high level description of the programmable logic device (PLD). These are the physical dimensions of the complex PLD. It also shows a very important aspect of the Nios board, the Avalon bus and it’s inter connects. It is the reason why the processor’s reconfigurations do not skew performance of the Nios board. The bottom right corner has a User-Defined port where u can create extra UARTs, timers or PIOs.

![Figure 6. The Programmable Logic Device.](image)

The Nios board and kit consists of the APEX 20K200E development board, 9-V DC power supply, a power cable, Serial Y cable, 6-foot 25-pin parallel port extension cable ByteBlasterMV cable and an LCD Module. The kit also comes with software packages to make full use of it.

The most significant of the aforementioned software packages are shown under Development Tools on the left in Figure 7 (shown below). The SOPC (system-on-a-chip) Builder is used to create new processors and the Quartus® Software is used to embed the processor and any other custom hardware modules. The SOPC builder consists of a synthesizer and simulator to create test and fit custom processors.
Figure 7. The Programmable Logic Device Block Diagram.

Figure 8. The Programmable Logic Device IC Layout.
The next few paragraphs will serve as an introduction to the Nios board and how it operates with emphasis placed on how it was utilized in the ViCoN project. The Apex device (APEX EP20K200EFC484) below the Flash (ext_flash) chip on the board is used to configure and evaluate an embedded Nios processor. The Nios Development board had two 64k x 16 SRAM devices, one 8-Mbit flash memory device and a SODIMM memory expansion socket. The ViCoN-Bot’s has only 1 megabyte of flash memory available and below is a break down.

<table>
<thead>
<tr>
<th>Flash Address Size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1C0000 – 0x1FFFFF</td>
<td>256 KByte Factory-default APEX configuration</td>
</tr>
<tr>
<td>0x180000 – 0x1BFFFF</td>
<td>256 KByte User-defined APEX configuration data</td>
</tr>
<tr>
<td>0x100000 – 0x17FFFF</td>
<td>512 KByte Nios instruction and nonvolatile data space</td>
</tr>
</tbody>
</table>

The MAX7064 device right of the flash (ext_flash) chip and right above the Apex device is a configuration controller. The configuration controller loads configuration data from the flash and clocks it into the APEX device. User-created configurations are loaded by shorting JP2 jumpers just below the 9 white switches on the board.

The Nios Development Board comes with a modular plug-in power supply. This 9 volt power supply plugs into connector J1 (top left of board) and supplies the input power for the on-board power management. The on-board power management produces the following three voltages: 3.3 volts—Used by the majority of the components on the board, including all APEX IO banks. 1.8 volts—Used as the core voltage for the APEX EP20K200E device. 5.0 volts—Provided for optional use with the 5V-tolerant Altera daughter card connectors. The ViCoN-Bot relies on the 5Volt pins available on JP9 and JP11.

The Nios Development Board contains one serial port connector for communications between the APEX EP20K200E and any external device. The serial port connects via a DB-9 type connector through an RS-232 transceiver. The board has a 6-pin header at location to assist in debugging the serial signals.

The Nios Development Board comes with a set of switches and LEDs which proved very useful. In debugging and testing. Some of the switches are dedicated to resetting the board or served as indicators of the board’s current status.
The most time saving feature and most utilized aspect of the Nios Board is its generation of a CPU SDK for any custom system you create. You can then create programs to control or run on the board. It supports its form of assembly, C and C++.

The ViCoN-Bot project relies heavily on communication between devices. The CMUcam from Carnegie Mellon University communicates by sending and receiving bytes of ASCII coded characters through a standard serial port. The Nios CPU development board responds similarly. The Nios board also generates and propagates pulse-width modulated signals to the servo direction and speed motors located in the Tamiya Hummer M1025.

The CMUcam has a serial port for sending bytes of data in response to queries to it. The initial design of the ViCoN-Bot took this into consideration and simply proposed the serial port as a point of communications between the camera and Nios board. Our initial test of this communication link yielded no positive results. Signals sent from either end were not reaching their destinations. This prompted a research into serial signal transmission and reception and into the architecture of serial ports and devices. The solution to the signal propagation problem was to be solved with a male-to-male serial null modem. The design and construction of this null modem will be discussed later, with specifications included in the robot design section.

ASCII encoded bytes were sent back and forth from the Nios Board to the Camera and the response displayed on the LCD module. The first ‘successful’ test with the serial null modem yielded gibberish on the LCD module provided with the Nios development kit. The cause of this was not easy to ascertain; thus, we assumed the origin of the problem to be from a baud rate mismatch between the Nios board and CMUcam. The camera’s default and ideal baud rate is 115200bps and is the same for the Nios board. To verify that 115200kbps was indeed the default and current setting on both devices, we connected the CMUcam and the Nios board to different PCs and created mock programs to simulate a Nios-Camera and Camera-Nios communication. There were no significant observations from these programs, which led us to create a new theory.

We considered the next most plausible cause: a high baud rate between the devices was not acceptable to either or both devices. The baud rate was lowered on the camera to its lowest of
9600bps; however, the operation of the camera had deteriorated, become error prone and very tedious to use. We also discovered the Nios development board's serial port's baud rate could not be altered in software. The baud rate could only be set alterable by recompiling a custom processor for the Nios board. This problem was compounded further with the lack of appropriate licenses to compile a custom processor.

In a bid to utilize time, a third scheme was devised in order to isolate the problem. This involved delaying commands and responses in software. During this time, we started receiving familiar ASCII characters, but not what was expected. This partially implied a possible operation at 115200bps. After doing extensive research on the software and devices, a feasible and workable problem was found. Our group discovered that the termination character expected on both devices differed. Our limited use of serial ports were based on PCs. PCs accept and use both termination characters. This led us to have two choices in reaching a solution: update the camera's firmware with a new termination character or have our programs on the Nios board conform to the camera’s standard and curb the use of standard library calls when communication with the camera was concerned. The latter presented a safer and more logical solution. Communication was now established at 115200bps. The next phase of the Nios-CMUcam interaction now relied on our overall software design.

Programmers of embedded systems do not have the luxury of almost limitless memory and processor speed their counterparts who program for complete systems of gigabytes of hard disk memory and hundreds of megabytes of random access memory world enjoy. We need to be accountable and very meticulous in our software designs. The code for the ViCoN-Bot incidentally needs to be low level enough to send and receive data at close to real time as possible.

For the sake of continuity and ease of use for the privileged students who will be continuing our work on the ViCoN-Bot, we decided to modularize and abstract as much of the code we deemed necessary and possible. A high level design of our software is depicted below:
vicon_camera.c

In Figure 9 above, the file vicon_camera.c is shown enclosed on the hardware plane CMUcam. Therefore, all communication between the CMUcam and the Nios Development board is located in vicon_camera.c. It handles all function calls to the camera taking into account latencies and necessary data buffering. The CMUcam comes equipped with a set of functions on its microcontroller. The functions take in 2-letter command (e.g. GV gets firmware version) or a 2-letter command with a set of numbers (eg TC 0 1 2 3 4 5 tracks colors based on ranges of colors given by the 6 numbers following it) and returns a set of ASCII coded characters. All but 2 functions available for the camera were implemented in vicon_camera. The main goal of this was to simplify the state machine code by abstracting and modularizing the code. Instead of dealing with every little nuance involved in retrieving some simple data like the camera’s firmware version, a simple call to the function GetVersion will respond with a simple response. All functional prototypes and global constants are located in vicon.h
**vicon_servo.c**

Vicon_servo.c encased in a block with the two motors (Servo Speed and Servo Direction) handles communication between the Nios board and the motors. Unlike vicon servo above it only needs to respond to input from the Nios board, more precisely from the state machine. It indirectly invokes 2 VHDL modules (Speed_control and Direction_Control will be discussed in the next section) which generate pulse width modulated signals for the direction motor and speed motor. The peripherals on the Nios board utilized in propagating the pulse width modulated signal will be discussed later with the custom processor created for the ViCoN-Bot.

**vicon.h**

Vicon.h holds all function prototypes, global constants, type definitions and structures. This includes specialized structures representing data packets received from the CMUcam. The CMUcam is capable of sending five different packet types, namely C, F, M, N and S packets. Each packet has two variations but only one was deemed useful.

Vicon.c will host the state machine that dictates what the ViCoN-Bot does and how it does it. The state machine located in vicon.c and the file itself will be discussed in another section. Nios.h holds all the necessary function prototypes, constructs and structures needed to control and or communicate with all hardware on the Nios board. A fairly complete C API is available, but our initial use of these functions proved detrimental to the ViCoN-Bot’s sensitivity. Two other files shown in the Figure 9 above are pio_lcd16207.c and pio_lcd16207.h. There hold all the necessary functions and structures necessary to import data from the Nios board to the LCD module. Initial modifications were undone to keep it standard for future versions.

In retrospect the entire package could have been done in one huge file but would render the project difficult to continue or alter. This way any group wishing to use the Nios board and the CMUcam will have a great resource, vicon_camera.c. Any change to the CMUcam would only warrant changes in vicon_camera.c. Similarly communication between the Nios board and the hummer is now ready for further enhancements. A switch to a single motor handling both speed and direction would also warrant changes only to the file vicon_servo.h.
Pulse Modulation with the Nios Board

Initially, a method of producing pulse width modulations (PWM) was to be determined. One method considered was to use a pulse width circuit to produce the signal. Another consideration was to use C or VHDL to program the Nios Board to produce pulse modulation. It was decided that it would be easier and cheaper to use VHDL to produce the PWM because of the amount of resources available on how produce PWM in VHDL.

The pulse width design contained two servo controllers in VHDL code named speed_control.vhd and direction_control.vhd, and a clock generator called clk_div.vhd. The files can be seen in Appendix C pages 61 through 66. Clk_div.vhd is driven by the Nios 33 MHz system clock which uses this clock signal to generate various clock speeds. Both servo controllers are designed to input three-bit numbers that corresponds to a respective speed or direction. The signals are driven by the 100 kHz clock that is produced by clk_div.vhd. Table C-1 illustrates how the two servo controls work. For the speed servo each three-bit value represents 8 settings ranging from stop or, brake, half reverse, neutral, and full forward. A neutral position is required in order to switch from forward motion to reverse and vice versa. The in between settings represent fractions of the top speeds desired. Each pulse is between one and two milliseconds long. The pulses have periodicity of twenty milliseconds; therefore, speed_control.vhd contains a counter set to count from 0 to 2000 incremented every clock cycle. The code tests for the pulse widths at the end of each clock cycle as shown in speed_control.vhd. Direction_control.vhd works the same way as speed_control.vhd except the three-bit number corresponds to eight direction settings ranging from a 45° left turn to a 45° right turn.

Symbols were generated for the three VHDL files after they were successfully compiled. The symbols were then connected to the system module as seen in Figure C-1. The output pins for the pulse modulation were on JP11 on the Nios board, shown in the Figure 10 below. The red boxes highlight which pins, on JP11, are used to connect to the servo motor pins on the robot. As shown in the figure pin R20 holds the speed servo three-bit number and pin R1 holds the direction servo three-bit number. Both of the ground pins were used and the external 5-volt regulator is used to power the servo.
After the whole design was successfully compiled, the pulse widths were tested using an oscilloscope. Figure C-2 and Figure C-4 show the oscilloscope screen captures of a 1.5 ms, 1.75 ms, and 2 ms pulse respectively. Examining these figures, along with the percent error calculations in Table C-1, illustrates how the actual pulses measured had significantly close duty cycles to the theoretical values that were calculated. It was concluded from the observations that the pulse widths were accurate enough to control the robot’s servo motors.

A C source file named vicon_servo.c, shown on pages 67 through 68 of Appendix C, is used to set the actual speed and direction. Vicon_servo.c contains functions that return an integer that represents a speed or direction setting. This code is also used in the state-machine to output this integer to the VHDL servo controllers which then produces the pulse width modulation.
Robot Design

The ViCoN-Bot was developed from the Tamiya M1025 4X4 Hummer RC hobbyist kit. The M1025 was equipped with a stock Mabuchi RS-540 27 turn carbon brush DC drive motor and upgraded with the Futaba MC330CR speed controller. Two 3-pin connectors on the robot chassis and speed controller are the input signals to the servo for front wheel steering and the speed controller. Red is the 5 volt reference wire, white is the PWM signal wire, and black is the ground reference wire.

Assembling the robot proved to be a cumbersome process. The Hummer came equipped with a removable top hatch which was useful for routing wires. First, the Nios board was mounted on back end of the Hummer. Holes were drilled in the proper locations and 4X40 screws were used to secure the Nios board to the body. Next, the LCD screen was mounted to the roof directly in front of the Nios board in a similar fashion. Figure E-1 shows the Nios board and LCD mounted to the Hummer. Thirdly, the CMUcam was mounted to the roof above the windshield using small aluminum L-brackets and secured using wing nuts for easy adjustment as shown in Figure E-2. The camera was aligned by iteratively dumping frames to the Java GUI and adjusting the camera positions till the image was centered properly. For an unknown reason, the mounted camera appeared drastically, visually misaligned; however, the image appeared centered in the bitmap despite the visual displacement.

The CMUcam was powered by four AA 1.5 volt batteries in a series arrangement providing the required 6 volts. The AA supply was mounted internally underneath the roof, and had a 2-pin connector that powered the CMUcam through the hatch opening. The M1025 drive motor and speed controller were powered with a 7.2 volt 2000mAh RC battery and stored in the Hummer’s built in battery compartment. The Nios board and front servo were powered with an identical 7.2 volt battery and mounted inside the body, directly underneath the Nios board using two tie wraps. Figure E-3 shows underneath the hood, the batteries, and chassis of the robot. Using different power sources for each major component isolated each unit from possible noise generated by the DC motor or other noise sources while providing extended battery lifetime. The camera and Nios board shared a common ground that was connected on pin 5 of the serial, null modem cable. The speed controller and servo had a common ground with the central Nios board through the ground wire in the 3-pin connectors. Connecting a common reference ground between each module was essential to proper electrical operation.
Initially, there was minimal wiring involved having only batteries and cables to connect. First, a null modem cable was created by soldering pins 2, 3, and 5 of two male RS-232 serial connectors as depicted in Figure 11.

![Figure 11. RS-232 Serial Null-Modem wiring diagram](image)

A power connector for the Nios board was then soldered to a connector to the battery and insulated with heat shrink wrap. It is important to note that the Nios power connector and the battery connector had different polarities and required that reverse wiring. Next, two twisted 4-pin computer connectors were used to connect the Nios board to the 3-pin signal wires inside the robot. The Nios board had only one 5 volt output pin and was used for the LCD screen leaving no 5 volt reference outputs for the PWM signal connectors. This was solved by constructing a longer ribbon cable with wires that extended beyond the connector on the Nios board. The second or red wire on the ribbon cable was determined to be the 5 volt wire. Cutting all the wires on the ribbon cable except the 5 volt wire left only the 5 volt wire exposed. The orange, 5 volt wire on the 4-pin connector and red 5 volt wire on the Nios board were then stripped, soldered together, and insulated with heat shrink wrap. The white wires on the 4-pin connectors were used as the signal wires. Since both connectors were identical, the 4-pin connector for speed was marked with a yellow dot on each end. The white signal wire of the speed connector was connected to pin 4 on the J11 of the Nios board, and the green ground wire was connected to pin 2. Likewise for direction, the white signal wire was connected to pin 40 of J11 and its green ground wire was connected to pin 36. Finally, all initial wiring was complete and testing was conducted.
After voltmeter and power up tests, all units appeared to be functioning properly as originally connected. However, as the ViCoN-Bot progress into the PWM stages, it was discovered that after powering the drives for a few minutes, the Nios board began flickering and behaving erratically until it shut off. The amount of current that the servo and speed control unit required exceeded the rated current of that pin and subsequently caused the Nios board to heat up until it approached failure. Solving this power limitation complicated the electrical design somewhat. It was decided that a separate circuit be built to provide the 5 volt reference potential needed by the servo and speed controller. A 5 volt, ½ amp voltage regulator, a 100 µF electrolytic capacitor, and a 10 pF polystyrene capacitor where used to construct this circuit. The two capacitors were put in parallel to decouple the noise induced due to conductive coupling both at high frequencies bands and low frequencies bands. By bypassing the Nios board with the voltage regulator circuit, the second battery now provided the PWM reference voltage directly. Since the Nios board and the voltage regulator share the same ground through a common source, there was no need for connecting grounds of the two circuits in any other way. The voltage regulator circuit was mounted about the back right tire, underneath the roof as partial depicted in Figure E-3. This new arrangement provided sufficient current when fully loaded by the servo and speed controller; whereas, the Nios board failed under the same conditions.

The ViCoN-Bot was constructed using the M1025 Hummer, Altera Nios board, CMUcam, several batteries, and a 5 volt regulatory circuit. Pictures shown in Figure E-4 and Figure E-5 depict a frontal and top view of the final ViCoN-Bot. During the testing phase, a styrofoam bummer, not shown in the figures, was attached to the front of the robot for protection against accidental collisions. The final hardware configuration and overall construction of the robot proved to be more than suitable for the ViCoN-Bot.

State Machine

One of the major milestones in the development of the ViCoN-Bot was complete communication, passing of information, between the camera and the Nios board. Without a consistent means of information there would be no sufficient way of accomplishing the overall goals presented in this design project. The CMUcam communicates through a series of packets containing pertinent information about the image that it is processing such as the color mean, coordinates of the tracked color, and size of an image (represented as a pixel value). The packets
from the camera are returned in two different ways the first of which being a constant stream of packets or a single packet at a time with a sufficient request to the camera. With the ability of communication between the camera and Nios board the development of a program to control the action of the ViCoN-Bot could be accomplished. A state machine served as this mechanism of control.

The state machine created for the robot serves as the internal “traffic-light” for the ViCoN-Bot. It controlled when packets where sent, how they were processed, and what signal was sent to the servos. **Figure 12** shows an outline of the state machine and how it transitions from state to state, making decisions for the robot.

![Figure 12. A representation of the state machine used for the ViCoN-Bot.](image)

The initial state, CameraStart, sets up the CMUcam for proper use. It initializes the camera functions of PollMode, MiddleMass, and ColorMode. The PollMode function is a necessity because it makes the camera send the packets one at a time. Ideally, a consistent stream of data would work better for detecting the position of an object, which would be 17 frames per second (fps) at the 112,500 baud rate used in the design, but the streaming data would be difficult to parse and use correctly. Setting the MiddleMass and ColorMode are done to specify the type of packets sent and to turn the cameras white balance on. The white balance is a color filter for improved image quality with present IR light.
The FindColor state serves as a means to identify if a color is present. It does a “round robin” check of each color until one of them is identified and then passes the color value on the next state, TrackingState. In the TrackingState most of the information gathering is done on the object being tracked by the robot. Average values are taking of the x position, y position, x middle mass, y middle mass, confidence (tells the confidence of the objects presence), and pixels (size in pixels of the object). Three packets of information have proven to be the optimal setting for taking the average values. If the average confidence is greater than a threshold value of 20, 255 is the highest value, then the ViCoN-Bot goes into a new state, DataControl, or it is just sent back to the FindColor state.

DataControl is where all the computation of the objects presence is made based on the collected information. The servo direction is calculated from the middle mass values returned. This value gives the location of the highest confidence for the center of the object being tracked. At the time of this writing this produces sufficient results on the direction to send the ViCoN servos but it will be necessary for a more robust direction calculation which will be discussed further in the testing section. Similarly, the speed setting is calculated with the pixel size and confidence but the speed of the robot is too great. Thus, the only usable speed is at the slowest setting because the damage done to the robot will be “unimaginable” at higher speed if there is a crash while testing. Figure 13 shows how the robot makes actual decisions based on the situation and the color of the object being tracked. A in Figure 13 is a typical situation where the robot would set the direction according to the data. B shows an example of how the camera would return to the FindColor state since pink is in the line of sight of the robot even though green is present. Pink serves as the dominant color and its presence supersedes the tracking of any color. C shows the camera tracking a yellow object.
There are also a Reset and Idle state that are initialized through the pushbuttons located on the Nios board. In theory, this is not a true state machine but more of top level control program to initiate and communicate with all the lower levels of code in the design. A major advantage to the program is the scalability it possess since additional colors to be tracked can be easily added (pink, green, and yellow were the original colors used). But the functioning state machine was a milestone in the ViCoN-Bot development. It was a milestone that unexpectedly took longer than planned because of a calculation error within the parsing of the incoming packets from the CMUcam. There was a delay that was made too short and the packets were not being properly received. With a baud rate set to 112,500 it is necessary to give the camera at least 400 ms to send a packet, because of the amount of characters returned, but when the delay is too short only a portion of the information is collected.

Testing

The testing of the ViCoN-Bot was the last phase of the project development. At the time of this paper 70-80% of the testing had been completed and the robot was fully functional. The testing had shown that the robot could navigate through a designed course of objects at set sizes and known distances. The lighting proved to also dramatically alter the results because the color readings can change to such a high extent. There would have to be a physical body there at all
times to keep the ViCoN-Bot on the course though if an object was out of its line of sight. Most of the testing was conducted in the College of Computing but ideally the robot will have much more success in the Van Leer because of the improved lighting and most of the original calculations were specified for that building. New calculations for the colors in the COC were compiled but further testing will be needed for optimization in both buildings.

Originally the testing course was optimized for the layout of the Van Leer. Figure 14 shows the layout of the proposed course within the building (second floor).

![Figure 14. Layout of the course to test the robot on the Van Leer second floor.](image)

Placed at each corner, would be a green poster board signaling when to initiate a left turn to the robot. The pink markers are hung in the middle of the hall, slightly higher than the ViCoN. Regular yarn was used to hold the sign up with regular pushpins. A similar set up was used in the COC by sectioning off one part of the hall and creating a smaller version of the course since
the hall dimensions are smaller there. Appendix D shows photographs of the track during the actual testing of the robot, see Figure D-1 and Figure D-2, in the COC.

The testing portion is an ongoing process and as previously mentioned is at roughly 75%. Many unexpected circumstances and new information about the functionality of the ViCoN has been unveiled through this process. The most important is the effect of shadows. In areas of bad illumination, such as the COC, the RGB values of a color can be drastically different. Figure 15 demonstrates how the color range of an object varies with overhead lighting. This effect limits the robustness of the tracking since color ranges must be expanded to properly track a color.

![Image of color range demonstration](image)

**Figure 15.** Example in the color range difference depending on available overhead lighting.

The next phase of testing will focus on creating a more diverse way of calculating the direction that produces better results in all lighting conditions. This should lead to a better functionality while the robot is on the course. Also, to improve on the left turns when green is the tracking color. Incorrect pixel (object size) values can through off the robots turn radius making it too short or long. This will need further investigation because a small amount of times during test runs of the course the ViCoN-Bot would start turning slightly early.
Organization

The ViCoN Design Project took place during the course of eleven weeks. Kwabena Bosompem, Jeff Vickers, Kevin Walker, and Andre Moore are the four members of the ViCoN team. There were two major milestones that had to be completed in order to complete the project within the allotted time. Completely interfacing the Nios Board and the CMUcam was the first milestone. This was to be completed by the end of week five. An operational robot was the second milestone that should have been done by the middle of week eight in order to finish the project. The Ghant chart in Figure 16 illustrates the major phases that make up the ViCoN Project. It also shows the duration of each phase with respect to the others.

![Ghant chart outlining ViCoN design structure.](image)

Research was the first major phase. This took place during the first two weeks of the project. During this time the project team searched for digital cameras and considered what programming device to use. When the digital camera was obtained and the programming
device was determined Jeff and Kevin tested the camera’s performance functionality, and limitations. Kwabena and Andre completed the Nios Tutorial.

The second phase involved Kwabena and Andre working to swiftly interface the CMUCam and the Nios Board. This involved sending commands to the camera via the Nios Board, receiving feedback from the camera, and printing it to the LCD. Kevin and Jeff designed a track and optimized the color ranges to be tracked.

Jeff and Kevin also worked on preparing the M1025 Hummer to interact with the CMUCam and the Nios Board, which was the bulk of the third phase. This phase included robot construction and installing power supplies. They also began designing a state-machine to control the motions of the robot. Andre and Kwabena were completing the software code that would interface all components of the ViCoN-project

Phase four dealt with the designing and implementing pulse width modulation. Jeff wrote the VHDL code to create pulse modulation for the direction and speed servos in the ViCoN-Bot. Andre and Kwabena worked on integrating the VHDL code in the top level design. Kevin began implementing the state machine into the source code. At the end of this phase the team achieved pulse width modulation and was able to control the servos. The robot was not fully operational by the end of week seven as planned but it was full operational by the middle of week eight.

Testing and tweaking was the major component of phase five as well as completing the project report, demonstration, and final presentation. Kevin completed the state-machine. All four team members worked to tweak and test the ViCoN-Bot on the track. The team made the final presentation and made a final demonstration.

Conclusion

Designing an autonomous, visually controlled robot, the ViCoN-Bot, encompassed nine weeks from concept to functional prototype. The goal of the ViCoN project was simply to create a vehicle that used a digital camera as a means of guidance around a course. The final prototype used an FPGA, a digital camera, and two motors mounted in a miniature Hummer.

Design and implementation of the ViCoN-Bot was modular, providing a basic framework for future prototypes to build upon. ViCoN’s modularity aids in a simplistic description of our accomplishments. The first major milestone was the successful communication between the
The second major milestone was the creation of a custom processor for the FPGA that incorporates modules necessary to generate signals to control the motors. The third and final major milestone was integrating the previous two and realizing the overall project.

The final phase involved positioning of brightly colored neon pink and green landmarks in a fairly well lit hallway much like traffic lights. Objects were placed a reasonable distance apart and elevated such that the ViCoN-Bot could pass underneath the object. The demo version was limited to the detection and tracking of two color ranges, fluorescent green and pink. Neon pink acted as a waypoint to seek and denoted forward movement. Similarly, neon green signified a left hand turn.

The CMUcam’s inherent limitations included a 25° field of view, a low resolution of 80x143, and a low refresh rate of 17 frames per second affecting the ViCoN-Bot’s overall speed and maneuverability. Moreover, the CMUcam’s imaging functions and algorithms were of an extremely basic nature limiting the degree to which further processing could be applied. The vehicle’s inability to spin or rotate about a point also limited the ability of the ViCoN-Bot to locate a direction or speed indicator.

The following are a few suggestions for continuing versions of the ViCoN-Bot or for groups interested in borrowing ViCoN modules. Altera has slated a new Nios board with a processor approaching three times the speed of the current one, allowing for more complex and speedy computations. Seattle Robotics and Carnegie Melon are also working on the next CMUcam which offers much higher resolution and speeds. Adding a pan and tilt unit to the camera will also help in finding objects.

Despite several problems encountered and subsequently surmounted we deem the creation of the ViCoN-Bot a great exercise in team work, software design, hardware design and prototyping of digital systems. The first prototype of the ViCoN-Bot successfully demonstrated the control of a robot with a digital camera, just one of the endless possibilities of this vision sensor.
Appendix A:

CMUcam Data, Images, & User Manual
Table A-1. Experimental tracking ranges for neon pink.

<table>
<thead>
<tr>
<th>Color</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 meters</td>
<td>Neon Pink Poster</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>Green</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Blue</td>
<td>14</td>
<td>20</td>
</tr>
<tr>
<td>6 meters</td>
<td>Neon Pink Poster</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>Green</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>Blue</td>
<td>16</td>
<td>38</td>
</tr>
<tr>
<td>9 meters</td>
<td>Neon Pink Poster</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>Green</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>Blue</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>12 meters</td>
<td>Neon Pink Poster</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>230</td>
<td>240</td>
</tr>
<tr>
<td>Green</td>
<td>220</td>
<td>240</td>
</tr>
<tr>
<td>Blue</td>
<td>14</td>
<td>25</td>
</tr>
<tr>
<td>Track</td>
<td>Neon Pink Poster</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>220</td>
<td>240</td>
</tr>
<tr>
<td>Green</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>Blue</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

Table A-2. Experimental tracking ranges for neon green.

<table>
<thead>
<tr>
<th>Color</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 meters</td>
<td>Neon Green Poster</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>98</td>
<td>120</td>
</tr>
<tr>
<td>Green</td>
<td>220</td>
<td>240</td>
</tr>
<tr>
<td>Blue</td>
<td>80</td>
<td>102</td>
</tr>
<tr>
<td>6 meters</td>
<td>Neon Green Poster</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>85</td>
<td>110</td>
</tr>
<tr>
<td>Green</td>
<td>180</td>
<td>215</td>
</tr>
<tr>
<td>Blue</td>
<td>50</td>
<td>75</td>
</tr>
<tr>
<td>9 meters</td>
<td>Neon Green Poster</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>75</td>
<td>90</td>
</tr>
<tr>
<td>Green</td>
<td>150</td>
<td>165</td>
</tr>
<tr>
<td>Blue</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>12 meters</td>
<td>Neon Green Poster</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>76</td>
<td>92</td>
</tr>
<tr>
<td>Green</td>
<td>140</td>
<td>160</td>
</tr>
<tr>
<td>Blue</td>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>Track</td>
<td>Neon Green Poster</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>75</td>
<td>120</td>
</tr>
<tr>
<td>Green</td>
<td>140</td>
<td>240</td>
</tr>
<tr>
<td>Blue</td>
<td>40</td>
<td>100</td>
</tr>
</tbody>
</table>
Table A-3. Experimental tracking ranges for neon yellow.

<table>
<thead>
<tr>
<th>Color</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 meters</td>
<td><strong>Neon Yellow Poster</strong></td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>Green</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>Blue</td>
<td>55</td>
<td>65</td>
</tr>
<tr>
<td>6 meters</td>
<td><strong>Neon Yellow Poster</strong></td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>Green</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>Blue</td>
<td>45</td>
<td>70</td>
</tr>
<tr>
<td>9 meters</td>
<td><strong>Neon Yellow Poster</strong></td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>Green</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>Blue</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>12 meters</td>
<td><strong>Neon Yellow Poster</strong></td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Green</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Blue</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Track</td>
<td><strong>Neon Yellow Poster</strong></td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>235</td>
<td>240</td>
</tr>
<tr>
<td>Green</td>
<td>235</td>
<td>240</td>
</tr>
<tr>
<td>Blue</td>
<td>40</td>
<td>80</td>
</tr>
</tbody>
</table>

Figure A-1. Tracking neon pink and green posters in EE hallway at 3, 6, 9, 12, and 15 meters.
Table A-4. Size and confidence measurements of a green poster at close proximity.

<table>
<thead>
<tr>
<th></th>
<th>Neon Green Poster</th>
<th></th>
<th>Neon Green Poster</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 meters</td>
<td></td>
<td>1.5 meters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>255</td>
<td>Size</td>
<td>189</td>
<td></td>
</tr>
<tr>
<td>Confidence</td>
<td>238</td>
<td>Confidence</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>2 meters</td>
<td></td>
<td>2.5 meters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>125</td>
<td>Size</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>Confidence</td>
<td>235</td>
<td>Confidence</td>
<td>228</td>
<td></td>
</tr>
<tr>
<td>3 meters</td>
<td></td>
<td>3.5 meters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>64</td>
<td>Size</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Confidence</td>
<td>230</td>
<td>Confidence</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>4 meters</td>
<td></td>
<td>4.5 meters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>40</td>
<td>Size</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Confidence</td>
<td>236</td>
<td>Confidence</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>5 meters</td>
<td></td>
<td>5.5 meters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>23</td>
<td>Size</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>Confidence</td>
<td>229</td>
<td>Confidence</td>
<td>202</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B:

Nios Board Interface Diagrams & C Code
```c
#include "nios.h"

// Universal Definitions
#define SUCCESS 1
#define FAILURE 0
#define YES 1
#define NO 0
#define TRUE 1
#define FALSE 0

/*-----------SPEEDS-------------*/
#define BRAKE 0
#define STOP 0
#define FULL_FORWARD 7
#define HALF_FORWARD 6
#define QUARTER_FORWARD 5
#define EIGHTH_FORWARD 4
#define NEUTRAL 3
#define QUARTER_REVERSE 2
#define EIGHTH_REVERSE 1

/*-----------DIRECTIONS-------------*/
#define STOP 0 //??
#define LEFT_10 1
#define LEFT_20 2
#define LEFT_45 3
#define LEFT_0 4
#define STRAIGHT 4
#define RIGHT_0 4
#define RIGHT_10 5
#define RIGHT_20 6
#define RIGHT_45 7

//Interrupt Context (what caused it)
typedef struct{  // Struct to hold pointer to uart struct
    np_uart *uart;  // Pointer to uart struct see nios.h
    char rxChar;    // Last byte sent through UART see nios.h
} UARTISRContext; // Named such to .... context of interrupt

typedef struct{
    int rmin;
    int rmax;
    int gmin;
    int gmax;
    int bmin;
    int bmax;
} Color;

static Color PINK = {210, 240, 0, 60, 10, 30};
static Color YELLOW = {225, 240, 225, 240, 30, 80};
static Color GREEN = {65, 120, 130, 240, 30, 100};

//Type C packet
typedef struct{
    int x1;  //The left most corner x value
    int y1;  //The left most corners y value
    int x2;  //The right most corners x value
    int y2;  //The right most corners y value
    int pixels;  //# of Pixels in the tracked region,
    int confidence;  //The (# of pixels / area)*256 of the
                     // bounded rectangle and capped at 255
} CdataPacket;
```
// Type M packet
typedef struct {
    int mx; // The middle of mass x value
    int my; // The middle of mass y value
    int x1; // The left most corners x value
    int y1; // The left most corners y value
    int x2; // The right most corners x value
    int y2; // The right most corners y value
    int pixels; // # of Pixels in the tracked region
    int confidence; // The (# of pixels / area)*256 of the // bounded rectangle and capped at 255
} MdataPacket;

// Type N packet
typedef struct {
    int spos; // The current servo position
    int mx; // The middle of mass x value
    int my; // The middle of mass y value
    int x1; // The left most corners x value
    int y1; // The left most corners y value
    int x2; // The right most corners x value
    int y2; // The right most corners y value
    int pixels; // # of Pixels in the tracked region
    int confidence; // The (# of pixels / area)*256 of the // bounded rectangle and capped at 255
} NdataPacket;

// Type S packet
typedef struct {
    int rmean; // the mean Red or Cr (approximates r-g)
    int gmean; // the mean Green or Y (approximates intensity)
    int bmean; // the mean Blue or Cb (approximates b-g)
    int rdev; // the *deviation of red or Cr
    int gdev; // the *deviation of green or Y
    int bdev; // the *deviation of blue or Cb
} SdataPacket;

/*----------FUNCTIONS PROTOTYPES--------------*/

// Vicon_Camera.c
int SendCommand(char *s);
int DisplayResponse(void);
int EmptyArray(void);
void ViconMagic(char *s);
void ViconMagic2(char *s);
int ViconTest(void);
int PacketTokenizer(void);
int UpdatePackets(void);
int SetCameraIdle(void);
int SetContrast (int contrastI);
int GetContrast(void);
int SetBrightness (int brightnessI);
int GetBrightness(void);
int SetColorMode (int colorMode);
int GetColorMode(void);
int SetClockSpeed(int clockspeedI);
int GetClockSpeed(void);
int SetExposure(int exposureI);
int GetExposure(void);
int ToggleExposure(void);
int DumpFrame(void);
int SetDelay(int delayI);
int GetDelay(void);
135 int GetMeanColor(void);
136 int GetVersion ( void );
137 int SetHalfHorizontalResolutionMode(int active);
138 int GetHalfHorizontalResolutionMode(void);
139 int GetServoInput(void);
140 int SetTrackingLight(int trackingI);
141 int SetLineMode(void);
142 int SetMiddleMass(int middlemassI);
143 int SetNoiseFilter(int noisefilterI);
144 int ToggleNoiseFilter(void);
145 int GetNoiseFilter ( void );
146 int SetPollMode(int pollmodeI);
147 int GetPollMode(void);
148 int SetRawMode(int rawmodeI);
149 int GetRawMode ( void );
150 int ResetCamera(void);
151 int SetServoPosition(int positionI);
152 int SetSwitchingMode(int switchingmodeI);
153 int GetSwitchingMode (void);
154 int SetWindowSize(int x, int y, int x1, int y1);
155 int TrackWindow(void);
156 int TrackColor(Color c);
157 void InterruptServiceRoutine(int context);
158 void DisableUartInterrupt(void);
159 void EnableUartInterrupt (void);
160 void InterruptHandler (int context);
161
162 // Vicon_Servo.c
163 int GetSpeed(void);
164 int GetDirection(void);
165 void SetSpeed(int spd);
166 void SetDirection(int drx);
167
168 // Vicon.c
169 int main2(void);
170 int FindColor(void);
171 int CameraStart(void);
172 int TrackState(void);
173 int DataControl(int con, int m1, int m2, int pix);
174 //int SendServo(ServoValue);
175 int Reset(void);
176 void MyPIO_ISR(int context);
177 void InitializeCamera(void);
178
179 // Vicon_pacet_test.c
180 int CheckPackets(void);
181
182
#include "vicon.h"
#include "nios.h"
#include "pio_lcd16207.h"
#include <stdio.h>
#include <string.h>

/*-----------------------------------------------------------------*
 * Author(s): Kwabena Asare Bosompem
 * Andre Moore
 * Jeff Vickers
 * Kevin Walker
 * Created: May 30th, 2002.
 * Description: Functions for Camera Control
 * Version: 0.1a
 * Comments: Every function that returns a data packet assumes
 * the poll mode is set to 1. Default camera setting
 * is 0. Also no raw mode. Not necessary for Nios.
 *-----------------------------------------------------------------*/

/*-----------------------------------------------------------------*
 * Definitions & Global Variables
 *-----------------------------------------------------------------*/

char lastresponse[100];
const char VER[18] = "ACK CMUcam v1.12 :";
const char RVER[19] = "ACK CMUcam v1.12 :"
const char ACK[5] = "ACK :");
const char NCK[5] = "NCK :");
const char COL[2] = ": ");
const char VER_RM[14] = "CMUcam v1.12 :");
const char RVER_RM[15] = " CMUcam v1.12 :");
const char ACK_RM[1] = ":");

const int AUTOGAIN_ON = 33; //DEFAULT
const int AUTOGAIN_OFF = 32;
const int YCC_ON = 36;
const int YCC_OFF = 32;
const int RGB_ON = 44;
const int RGB_OFF = 40;
const int TRACKING_ON = 1;
const int TRACKING_OFF = 0;
const int TRACKING_AUTO = 2;

int crindex = 0; // Keep track of camera response array
UARTISRContext gC = {0,-1}; // Context and last byte preset.
int k=0; // Index of array to be tokenized for packets;
int crtemp=0;
int temp=0;
SdataPacket spacket = {0};
MdataPacket mpacket = {0};
NdataPacket npacket = {0};
CdataPacket cpacket = {0};

int pollmode = 0; // Initialize poll mode
int rawmode = 2; // Initialize raw mode
int mmassmode = 1; // Initialize middle mass mode
int uartstatus = 0; // Status of serial port
int clockspeed = 17; // Default values
int contrast = 127;
int brightness = 127;
int exposure = 1;
int delay = 1;
int colormode = 40;
int active = 0;
int tracking = 2;
int middlemass = 1;
int noisefilter = 1;
int position = 0;
int switchingmode = 0;

/*-----------------------------------------------------------------*/
FUNCTIONS
*/

/**
SEND COMMAND (STRING)
Description: Sends through the uart(serial port) a preformatted command(string) to the CMUcam.
Created: June 6th, 2002.
Parameters: Pointer to a string(s)
Returns: Int indicating success or failure
Comments: See nios_peripheral_reference_manual.pdf for details on nr_uart_txstring
Modified: June 14th, 2002.
*/
int SendCommand(char *s)
{
    uartstatus = nr_uart_txstring(s);
    if (uartstatus > 0)
        return SUCCESS;
    else
        return FAILURE;
}

/**
DISPLAY RESPONSE()
Description: Displays to the LCD the last string recieved from the CMUcam.
Created: June 6th, 2002.
Parameters: Void
Returns: int to indicate successful print to lCd
Comments: Took delay out. Delays need to be added in main.
Modified: June 15th, 2002.
*/
int DisplayResponse(void)
{
    nr_pio_lcdwritescreen(lastresponse);
    return SUCCESS;
}

/**
EMPTY ARRAY()
Description: Empties the array holding a string of the last camera response
Created: June 6th, 2002.
Parameters: Void
Returns: int
Comments: Currently looking for quicker way to achieve this with pointers
Modified: June 14th, 2002.
int EmptyArray()
{
    int i;
    for (i=0; i<100; i++)
    {
        lastresponse[i]= 0;  // 0 is ascii for null
    }
    return SUCCESS;
}

void ViconMagic(char *s)
{
    crindex = 0;
    k = 0;
    EmptyArray();
    EnableUartInterrupt();
    SendCommand(s);
    nr_delay(100);
    DisableUartInterrupt();
}

void ViconMagic2(char *s)
//The packets need extra time in a delay since it is more characters returned.
//Without the 1/2 ms delay the packets would be clipped.
{
    crindex = 0;
    k = 0;
    EmptyArray();
    EnableUartInterrupt();
    SendCommand(s);
    nr_delay(500);
    DisableUartInterrupt();
}

int ViconTest(void)
{
    int rmode;
    rmode = GetRawMode();
    rmode = rmode & 2;
    if (rmode == 2)
        if (strcmp(ACK, lastresponse) != 0)
            return FAILURE;
    else
return SUCCESS;
else
    if (strcmp(ACK_RM,lastresponse)!=0)
        return FAILURE;
    else
        return SUCCESS;

}//end of ViconTest

/**
 * PACKET TOKENIZER
 * Description: Dumps bitmap of current still image
 * Created: June 13th, 2002.
 * Paramaters: Void
 * Returns: Int
 * Comments: Splits into consistuent numbers
 * Modified: June 7th, 2002.
 */

int PacketTokenizer(void){
    int xi;
    while(lastresponse[++k]== 32);
    crtemp =lastresponse[k];
    while((crtemp<58) && (crtemp>47)){
        temp = temp*10 + crtemp - 48;
        k++;
        crtemp =lastresponse[k];
    } //end of
    xi = temp;
    temp = 0;
    return xi;
}

}//end of PacketTokenizer()

/**
 * UPDATE PACKETS (VOID)
 * Description: Dumps bitmap of current still image
 * Created: June 13th, 2002.
 * Paramaters: Void
 * Returns: Int
 * Comments: Splits into consistuent numbers
 * Modified: June 7th, 2002.
 */

int UpdatePackets(void){
    while((k<100) || (lastresponse[k] == 0)){
        if(lastresponse[k] == 65 ) // 65 is ascii A
            k++;
        k++;
        if(lastresponse[k] == 83 ) // 83 is ascii S
            {
                spacket.rmean = PacketTokenizer();
                spacket.gmean = PacketTokenizer();
                spacket.bmean = PacketTokenizer();
                spacket.rdev = PacketTokenizer();
                spacket.gdev = PacketTokenizer();
                spacket.bdev = PacketTokenizer();
            }
        if(lastresponse[k] == 67 ) // 67 is ascii C
            {
                cpacket.x1 = PacketTokenizer();
                cpacket.y1 = PacketTokenizer();
            }
    }
}
cpacket.x2 = PacketTokenizer();
cpacket.y2 = PacketTokenizer();
cpacket.pixels = PacketTokenizer();
cpacket.confidence = PacketTokenizer();
}
if(lastresponse[k] == 77 ) // 77 is ascii M
{
  mpacket.mx = PacketTokenizer();
  mpacket.my = PacketTokenizer();
  mpacket.x1 = PacketTokenizer();
  mpacket.y1 = PacketTokenizer();
  mpacket.x2 = PacketTokenizer();
  mpacket.y2 = PacketTokenizer();
  mpacket.pixels = PacketTokenizer();
  mpacket.confidence = PacketTokenizer();
}
if(lastresponse[k] == 78 ) // 78 is ascii N
{
  npacket.mx = PacketTokenizer();
  npacket.my = PacketTokenizer();
  npacket.x1 = PacketTokenizer();
  npacket.y1 = PacketTokenizer();
  npacket.x2 = PacketTokenizer();
  npacket.y2 = PacketTokenizer();
  npacket.pixels = PacketTokenizer();
  npacket.confidence = PacketTokenizer();
}
k++;
return SUCCESS;

int SetCameraIdle (void)
{
  char cmdtosend [4] = "\r";
  ViconMagic(cmdtosend);
  return ViconTest();
}//end of SetCameraIdle()

/* Set Camera Idle ()
   Description: Sets Camera Idle. Stops streaming of packets in
   when camera is in tracking mode.
   * Created: June 7th, 2002.
   * Parameters: Void
   * Returns: Int SUCCESS or FAILURE
   * Comments:
   * Modified: June 15th, 2002 */
int SetCameraIdle (void)
{
  char cmdtosend [4] = "\r";
  ViconMagic(cmdtosend);
  return ViconTest();
}//end of SetCameraIdle()

/* Set Contrast ( Int )
   Description: Set's the camera's contrast
   * Created: June 15th, 2002.
   * Parameters: contrastI. an int from 0-255
   * Returns: int (SUCCESS or FAILURE)
   * Comments: */
/* Modified: June 15th, 2002. */

int SetContrast (int contrastI)
{
  char cmdtosend [6];
  contrast = contrastI;
  if((contrast >255) || (contrast < 0))
    return FAILURE;
  sprintf(cmdtosend, "CR 5 %d\r", contrast);
  ViconMagic(cmdtosend);
  return ViconTest();
}

int GetContrast (void)
{
  return contrast;
}

int SetBrightness (int brightnessI)
{
  char cmdtosend [6];
  brightness = brightnessI;
  if((brightness >255) || (brightness < 0))
    return FAILURE;
  sprintf(cmdtosend, "CR 6 %d\r", brightness);
  ViconMagic(cmdtosend);
  return ViconTest();
}

int GetBrightness (void)
{
  return brightness;
}

int SetColorMode (int colormode)
{
  char cmdtosend [6];
  if((colormode >44) || (colormode < 32)) //inadequate check
    return FAILURE;
  sprintf(cmdtosend, "CR 18 %d\r", colormode);
  ViconMagic(cmdtosend);
  return ViconTest();
}
```c
int SetClockSpeed(int clockspeedI)
{
    char cmdtosend [6];
    if((clockspeedI>12) || (clockspeedI< 2)) //inadequate check
        return FAILURE ;
    clockspeed = clockspeedI;
    sprintf(cmdtosend, "CR 17 %d\r", colormode);
    ViconMagic(cmdtosend);
    return ViconTest();
}

int GetClockSpeed(void)
{
    return clockspeed;
}

int SetExposure(int exposureI)
{
    char cmdtosend [6];
    if(exposureI != AUTOGAIN_ON || exposureI != AUTOGAIN_OFF)
        return FAILURE ;
    exposure = exposureI;
    sprintf(cmdtosend, "CR 19 %d\r", exposure);
    ViconMagic(cmdtosend);
    return ViconTest();
}

int ToggleExposure (void)
```
int status;
if(exposure == AUTOGAIN_ON)
    status = SetExposure(AUTOGAIN_OFF);
else
    status = SetExposure(AUTOGAIN_ON);
return status;
}

int DumpFrame(void)
{
    char command[5] = "DF\r";
    //Not to be used
    //ViconMagic(command);
    nr_pio_lcdwritescreen("Gee! what were you thinking");
    nr_delay(100000); //10 Sec wait. Same time it takes to dump frame
    return SUCCESS;
}

int SetDelay(int delayI)
{
    char cmdtosend[6];
    if((delayI>255) || (delayI<0)) //inadequate check
        return FAILURE;
    delay = delayI;
    sprintf(cmdtosend, "DM %d\r", delay);
    ViconMagic(cmdtosend);
    return ViconTest();
}

int GetDelay(void)
{
    return delay;
}
int GetMeanColor(void)
{
    char cmdtosend[4] = "GM\r";
    ViconMagic(cmdtosend);
    UpdatePackets();
    return SUCCESS;
}

int GetVersion(void)
{
    char command[5] = "GV\r";
    int rmode;
    ViconMagic(command);
    rmode = GetRawMode();
    rmode = rmode & 2;
    if (rmode == 2)
        if (strcmp(VER, lastresponse) != 0)
            return FAILURE;
        else
            return SUCCESS;
    else
        if (strcmp(VER_RM, lastresponse) != 0)
            return FAILURE;
        else
            return SUCCESS;
}

int GetHalfHorizontalResolution(int)
{
    char command[5] = "0D\r";
    ViconMagic(command);
    return SUCCESS;
}
int SetHalfHorizontalResolutionMode(int active)
{
    //active = 1 => every odd column processed
    //active = 0 => disable this mode
    nr_pio_lcdwritesscreen("HM ? Why oh why");
    nr_delay(100000);  //Wait 10 secs
    return SUCCESS;
}

int GetHalfHorizontalResolutionMode(void)
{
    return active;
}

/**
 * GET SERVO INPUT
 * Description: Gets the voltage on the servo line.
 * Created: June 15th, 2002.
 * Parameters: None
 * Returns: int -1 failure (0 low voltage or 1 high voltage)
 * Comments:
 */
int GetServoInput(void)
{
    char cmdtosend[4] = "I1\r";
    int pos;
    int rmode;
    ViconMagic(cmdtosend);
    rmode = GetRawMode();
    rmode = rmode & 2;
    if (rmode == 2)
        pos = 0;
    else
        pos = 4;
    if (lastresponse[pos] == 48)
        return 0;
    else if (lastresponse[pos] == 49)
        return 1;
    else
        return -1;
}

/**
 * SET TRACKING LIGHT ( INT )
 * Description: Controls the tracking light.
 * Acceptable values 0,1,2(default)
 * Created: June 15th, 2002.
 * Parameters: trackingI
 * Returns: int (SUCCESS or FAILURE )
 * Comments:
 */
int SetTrackingLight(int trackingI)
{
    char cmdtosend [6];
    if((trackingI >2) || (trackingI < 0))
        return FAILURE ;
    tracking = trackingI;
    sprintf(cmdtosend, "L1 %d\r", tracking);
    ViconMagic(cmdtosend);
    return ViconTest();
}

int SetLineMode(void)
{
    return SUCCESS;
}

int SetMiddleMass(int middlemassI)
{
    char cmdtosend [6];
    //if((middlemassI >10) || (middlemassI < 0))
    //    return FAILURE ;
    middlemass = middlemassI;
    sprintf(cmdtosend, "MM %d\r", middlemass);
    ViconMagic(cmdtosend);
    return ViconTest();
int SetNoiseFilter(int noisefilterI)
{
    char cmdtosend[6];
    if((noisefilterI > 2) || (noisefilterI < 0))
        return FAILURE;
    noisefilter = noisefilterI;
    sprintf(cmdtosend, "NF %d\r", noisefilter);
    ViconMagic(cmdtosend);
    return ViconTest();
}

int ToggleNoiseFilter(void)
{
    if(noisefilter == 0)
        return SetNoiseFilter(1);
    else
        return SetNoiseFilter(0);
}

int GetNoiseFilter(void)
{
    return noisefilter;
}

int SetPollMode(int pollmodeI)
{
    char cmdtosend[6];
    //if((pollmodeI == 0) || (pollmodeI == 1))
    //    return FAILURE;
    pollmode = pollmodeI;
    sprintf(cmdtosend, "PM %d\r", pollmode);
    ViconMagic(cmdtosend);
    DisplayResponse();
    nr_delay(800);
return ViconTest();

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}

int GetPollMode(void)
{
    return pollmode;
}

int GetPollMode(int)
{
    return ViconTest();
}
return SUCCESS;
else
    if (strcmp(RVER_RM,lastresponse)!=0)
        return FAILURE;
    else
        return SUCCESS;

}//end of ResetCamera

/**
 * SET SERVO POSITION (INT)
 * Description: Set's the servo position
 * Created: June 15th, 2002.
 * Parameters: position 0-255
 * Returns: int (SUCCESS or FAILURE)
 * Comments:
 */
int SetServoPosition(int positionI)
{
    char cmdtosend [5];
    //if((positionI > 255) || (positionI < 0)) //avoids other modes
    // return FAILURE;
    position = positionI;
    sprintf(cmdtosend,"S1 %d\r",position);
    ViconMagic(cmdtosend);
    return ViconTest();
}

}//end of SetServoPosition()

/**
 * SET SWITCHING MODE (INT)
 * Description: Switch between color and statistic packets
 * in tracking modes. 0- off 1-on
 * Created: June 15th, 2002.
 * Parameters: switchingmodeI
 * Returns: int (SUCCESS or FAILURE)
 * Comments:
 */
int SetSwitchingMode(int switchingmodeI )
{
    char cmdtosend [5];
    //if(switchingmodeI != 0 || switchingmodeI != 1) //avoids other modes
    // return FAILURE;
    switchingmode = switchingmodeI;
    sprintf(cmdtosend,"SM %d\r",switchingmode);
    ViconMagic(cmdtosend);
    return ViconTest();
}

}//end of SetSwitchingMode

int GetSwitchingMode (void)
{
    return switchingmode;

}//end of GetSwitchingMode
* SET WINDOW SIZE (INT INT INT INT)
* Description: Simply sets the window size of the camera
* Created: June 15th, 2002.
* Parameters: 2 coordinates top-left bottom-right
* Returns: int (SUCCESS or FAILURE)
* Comments:

```c
int SetWindowSize(int x, int y, int x1, int y1)
{
    char cmdtosend[20];
    if((x>x1 || y>y1) || ((x<1 || y<1) || (y1>143 || x1>80)))
        return FAILURE;
    sprintf(cmdtosend, "SW %d %d %d %d\r", x, y, x1, y1);
    ViconMagic(cmdtosend);
    return ViconTest();
}
```

---

* TRACK WINDOW()
* Description: Tracks prominent color in window
* Created: June 5th, 2002.
* Parameters: Void
* Returns: int
* Comments: Not to be used. Not beneficial for ViCoN-Bot

```c
int TrackWindow(void)
{
    char cmdtosend[3] = "TW\r";
    ViconMagic(cmdtosend);
    return UpdatePackets();
}
```

---

* TRACK COLOR (INT)
* Description: Tracks a given color
* Created: June 5th, 2002.
* Parameters:
* Returns:
* Comments:

```c
int TrackColor(Color c) /*array of color values*/
{
    char cmdtosend[20];
    //need to check array size and valid values
    sprintf(cmdtosend, "TC %d %d %d %d %d \r", c.rmin, c.rmax, c.gmin, c.gmax, c.bmin, c.bmax);
```
1005   ViconMagic2(cmdtosend);
1006   UpdatePackets();
1007   //return UpdatePackets();
1008
1009 }//end of TrackColor
1010
1011
1012 /*****************************************************************************/
1013 /*************************[ THE END ]*******************************************/
1014 /*****************************************************************************/
1015 /***************************************************************************/
1016 /**
1017 * INTERRUPT SERVICE ROUTINE()
1018 * Description: Handles all interrupt service routines
1019 * Created: June 5th, 2002.
1020 * Paramaters: int
1021 * Comments: Not to be used. Not beneficial for ViCoN-Bot
1022 * Modified: June 11th, 2002.
1023 */
1024 void InterruptServiceRoutine(int context)
1025 {
1026    UARTISRContext *c = (UARTISRContext *)context;
1027    int status;
1028    int rxChar;
1029    //char joy;
1030    status = c->uart->np_uartstatus;
1031    rxChar = c->uart->np_uartrxdata;
1032    c->uart->np_uartstatus = 0; // clear the interrupt condition
1033    if(status & np_uartstatus_rrdy_mask ){
1034       if(rxChar == 13)
1035          rxChar = 32;
1036       if(crindex>=0)
1037           lastresponse[crindex] = (char)rxChar;
1038       //joy = (char)rxChar;
1039       //lastresponse[crindex] = joy;
1040       crindex++;
1041    }
1042    c->rxChar = rxChar; // save the character for later use.
1043
1044 }//end of InterruptServiceRoutine
1045
1046
1047 /*****************************************************************************/
1048 /**
1049 * ENABLE UART INTERRUPT()
1050 * Description: Enables Uart as an interrupt
1051 * Created: June 5th, 2002.
1052 * Paramaters: int
1053 * Returns: void
1054 * Comments:  
1055 * Modified: June 7th, 2002.
1056 */
1057 void EnableUartInterrupt(void)
1058 {
1059    gC.uart = (np_uart *)na_uart1;
1060    nr_installuserisr(na_uart1_irq,InterruptServiceRoutine,(long)&gC);
1061    {
1062       np_uart *uart;
1063       uart = na_uart1;
1064       uart->np_uartcontrol = np_uartcontrol_irrdy_mask;
1065    }//end of thread. don't understand this
1066
1067 }//end of EnableUartInterrupt()
DISABLE UART INTERRUPT()
Description: Disables Uart as an interrupt
Created: June 5th, 2002.
Parameters: int
Returns: void
Comments: Modified: June 7th, 2002.*

```c
void DisableUartInterrupt(void)
{
    np_uart *uart;
    uart = na_uart1;
    uart->np_uartcontrol = 0;
    nr_installuserisr(na_uart1_irq, 0, 0);
}
```

//end of DisableUartInterrupt()
Appendix C:

Pulse Width Modulation Calculations & Code
Table C-1. Pulse width modulation calculations and settings.

<table>
<thead>
<tr>
<th>Clock (kHz)</th>
<th>Base Pulse (ms)</th>
<th># of clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>104.20</td>
<td>1</td>
<td>104</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Desired Speed</th>
<th>Speed</th>
<th>Setting #</th>
<th>Pulse Width</th>
<th>Duty Cycle</th>
<th>Duty Cycle Error</th>
<th>3-bit Value</th>
<th># of Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Stop / Brake</td>
<td>0</td>
<td>0</td>
<td>0%</td>
<td>0%</td>
<td>000</td>
<td>N/A</td>
</tr>
<tr>
<td>- 1/4</td>
<td>Quarter Reverse</td>
<td>2</td>
<td>1.375</td>
<td>7.164%</td>
<td>0.025%</td>
<td>010</td>
<td>1843</td>
</tr>
<tr>
<td>- 1/8</td>
<td>Eighth Reverse</td>
<td>1</td>
<td>1.438</td>
<td>7.489%</td>
<td>-0.013%</td>
<td>001</td>
<td>1850</td>
</tr>
<tr>
<td>0</td>
<td>Neutral</td>
<td>3</td>
<td>1.500</td>
<td>7.815%</td>
<td>0.008%</td>
<td>011</td>
<td>1856</td>
</tr>
<tr>
<td>1/8</td>
<td>Eighth Forward</td>
<td>4</td>
<td>1.563</td>
<td>8.141%</td>
<td>0.012%</td>
<td>100</td>
<td>1863</td>
</tr>
<tr>
<td>1/4</td>
<td>Quarter Forward</td>
<td>5</td>
<td>1.625</td>
<td>8.466%</td>
<td>0.015%</td>
<td>101</td>
<td>1869</td>
</tr>
<tr>
<td>1/2</td>
<td>Half Reverse</td>
<td>6</td>
<td>1.750</td>
<td>9.118%</td>
<td>0.007%</td>
<td>110</td>
<td>1882</td>
</tr>
<tr>
<td>1</td>
<td>Full Forward</td>
<td>7</td>
<td>2.000</td>
<td>10.420%</td>
<td>0.010%</td>
<td>111</td>
<td>1908</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Desired Angle</th>
<th>Angular Direction</th>
<th>Setting #</th>
<th>Pulse Width</th>
<th>Duty Cycle</th>
<th>Duty Cycle Error</th>
<th>3-bit Value</th>
<th># of Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>No Signal</td>
<td>0</td>
<td>0</td>
<td>0%</td>
<td>0%</td>
<td>000</td>
<td>N/A</td>
</tr>
<tr>
<td>-45</td>
<td>Left 45°</td>
<td>3</td>
<td>1.250</td>
<td>6.513%</td>
<td>-0.002%</td>
<td>011</td>
<td>1830</td>
</tr>
<tr>
<td>-20</td>
<td>Left 20°</td>
<td>2</td>
<td>1.389</td>
<td>7.236%</td>
<td>-0.009%</td>
<td>010</td>
<td>1845</td>
</tr>
<tr>
<td>-10</td>
<td>Left 10°</td>
<td>1</td>
<td>1.444</td>
<td>7.526%</td>
<td>-0.030%</td>
<td>001</td>
<td>1851</td>
</tr>
<tr>
<td>0</td>
<td>Home Position</td>
<td>4</td>
<td>1.500</td>
<td>7.815%</td>
<td>-0.002%</td>
<td>100</td>
<td>1856</td>
</tr>
<tr>
<td>10</td>
<td>Right 10°</td>
<td>5</td>
<td>1.556</td>
<td>8.104%</td>
<td>0.014%</td>
<td>101</td>
<td>1862</td>
</tr>
<tr>
<td>20</td>
<td>Right 20°</td>
<td>6</td>
<td>1.611</td>
<td>8.394%</td>
<td>0.006%</td>
<td>110</td>
<td>1868</td>
</tr>
<tr>
<td>45</td>
<td>Right 45°</td>
<td>7</td>
<td>1.750</td>
<td>9.118%</td>
<td>0.001%</td>
<td>111</td>
<td>1882</td>
</tr>
</tbody>
</table>
Figure C-1. Overall VHDL schematic including servo controllers and clock.
Figure C-2. Oscilloscope screen capture of a 1.5 ms pulse train every 20 ms.

Figure C-3. Oscilloscope screen capture of a 1.75 ms pulse train every 20 ms.
Figure C-4. Oscilloscope screen capture of a 2 ms pulse train every 20 ms.
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY clk_div IS
  PORT
  ( clock_33Mhz : IN STD_LOGIC;
    clock_1MHz : OUT STD_LOGIC;
    clock_100KHz : OUT STD_LOGIC;
    clock_10KHz : OUT STD_LOGIC;
    clock_1KHz : OUT STD_LOGIC;
    clock_100Hz : OUT STD_LOGIC;
    clock_10Hz : OUT STD_LOGIC;
    clock_1Hz : OUT STD_LOGIC);
END clk_div;

ARCHITECTURE a OF clk_div IS
  SIGNAL count_1Mhz: STD_LOGIC_VECTOR (4 DOWNTO 0);
  SIGNAL count_100Khz, count_10Khz, count_1Khz : STD_LOGIC_VECTOR (2 DOWNTO 0);
  SIGNAL count_100hz, count_10hz, count_1hz : STD_LOGIC_VECTOR (2 DOWNTO 0);
  SIGNAL clock_1Mhz_int , clock_100Khz_int , clock_10Khz_int , clock_1Khz_int :
    STD_LOGIC;
  SIGNAL clock_100hz_int , clock_10Hz_int , clock_1Hz_int : STD_LOGIC;
BEGIN
  PROCESS
  BEGIN
    -- Divide by 25
    WAIT UNTIL clock_33Mhz'EVENT and clock_33Mhz = '1';
    IF count_1Mhz < 32 THEN
      count_1Mhz <= count_1Mhz + 1;
    ELSE
      count_1Mhz <= "00000";
    END IF;
    IF count_1Mhz < 16 THEN
      clock_1Mhz_int <= '0';
    END IF;
    END PROCESS;

    -- Ripple clocks are used in this code to save prescalar hardware
    -- Sync all clock prescalar outputs back to master clock signal
    clock_1Mhz <= clock_1Mhz_int;
    clock_100Khz <= clock_100Khz_int;
    clock_10Khz <= clock_10Khz_int;
    clock_1Khz <= clock_1Khz_int;
    clock_100hz <= clock_100hz_int;
    clock_10hz <= clock_10hz_int;
    clock_1hz <= clock_1hz_int;
    END PROCESS;

  -- Divide by 10
  PROCESS
  BEGIN
    WAIT UNTIL clock_1Mhz_int'EVENT and clock_1Mhz_int = '1';
    IF count_100Khz /= 4 THEN
      count_100Khz <= count_100Khz + 1;
    ELSE
      count_100Khz <= "000";
      clock_100Khz_int <= NOT clock_100Khz_int;
    END IF;
    END PROCESS;

END clk_div;
-- Divide by 10
PROCESS
BEGIN
  WAIT UNTIL clock_100Khz_int'EVENT and clock_100Khz_int = '1';
  IF count_10Khz /= 4 THEN
    count_10Khz <= count_10Khz + 1;
  ELSE
    count_10Khz <= "000";
    clock_10Khz_int <= NOT clock_10Khz_int;
  END IF;
END PROCESS;

-- Divide by 10
PROCESS
BEGIN
  WAIT UNTIL clock_10Khz_int'EVENT and clock_10Khz_int = '1';
  IF count_1Khz /= 4 THEN
    count_1Khz <= count_1Khz + 1;
  ELSE
    count_1Khz <= "000";
    clock_1Khz_int <= NOT clock_1Khz_int;
  END IF;
END PROCESS;

-- Divide by 10
PROCESS
BEGIN
  WAIT UNTIL clock_1Khz_int'EVENT and clock_1Khz_int = '1';
  IF count_100hz /= 4 THEN
    count_100hz <= count_100hz + 1;
  ELSE
    count_100hz <= "000";
    clock_100hz_int <= NOT clock_100hz_int;
  END IF;
END PROCESS;

-- Divide by 10
PROCESS
BEGIN
  WAIT UNTIL clock_100hz_int'EVENT and clock_100hz_int = '1';
  IF count_10hz /= 4 THEN
    count_10hz <= count_10hz + 1;
  ELSE
    count_10hz <= "000";
    clock_10hz_int <= NOT clock_10hz_int;
  END IF;
END PROCESS;

END a;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY speed_control IS
    PORT (clock_100kHz : IN STD_LOGIC;
          speed : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
          speed_pwm : OUT STD_LOGIC);
END speed_control;

ARCHITECTURE servo_controller OF speed_control IS
    SIGNAL count_motor: STD_LOGIC_VECTOR (10 DOWNTO 0);
BEGIN
    PROCESS
    BEGIN
        -- Count_motor is a 20ms timer => [0, 2000] based on a true 100kHz clock --
        WAIT UNTIL clock_100kHz'EVENT AND clock_100kHz = '1';
        IF count_motor /= 2000 THEN
            count_motor <= count_motor + 1;
        ELSE
            count_motor <= "00000000000";
        END IF;

        -- Speed Setting #1 of 8 => BRAKE or OFF = 000 --
        IF count_motor >= 1700 AND count_motor < 1838 THEN
            IF speed = "000" THEN
                speed_pwm <= '0';
            ELSE
                speed_pwm <= '1';
            END IF;

        -- At this point the pulse is already 1 ms long --
        -- Decide how long to allow pulse to remain high. [1ms, 2ms] --
        ELSIF count_motor >= 1838 AND count_motor < 1844 THEN
            CASE speed IS
            WHEN "010" =>
                speed_pwm <= '0';
            WHEN "001" =>
                speed_pwm <= '1';
            WHEN "011" =>
                speed_pwm <= '1';
            WHEN "100" =>
                speed_pwm <= '1';
            WHEN "101" =>
                speed_pwm <= '1';
            WHEN "110" =>
                speed_pwm <= '1';
            WHEN "111" =>
                speed_pwm <= '1';
            WHEN OTHERS => NULL;
            END CASE;
            ELSE
                speed_pwm <= '0';
            END IF;

        -- Speed Setting #3 of 8 => Eighth Reverse (CW) --
        ELSIF count_motor >= 1844 AND count_motor < 1850 THEN
            WHEN "010" =>
                speed_pwm <= '0';
            WHEN "001" =>
                speed_pwm <= '1';
            WHEN "011" =>
                speed_pwm <= '1';
            WHEN "100" =>
                speed_pwm <= '1';
            WHEN "101" =>
                speed_pwm <= '1';
            WHEN "110" =>
                speed_pwm <= '1';
            WHEN "111" =>
                speed_pwm <= '1';
            WHEN OTHERS => NULL;
            END CASE;
            ELSE
                speed_pwm <= '0';
            END IF;

        -- Speed Setting #3 of 8 => Eighth Reverse (CW) --
        ELSIF count_motor >= 1844 AND count_motor < 1850 THEN
IF speed /="000" THEN
  CASE speed IS
  WHEN "010" =>
    speed_pwm <= '0';
  WHEN "001" =>
    speed_pwm <= '0';
  WHEN "011" =>
    speed_pwm <= '1';
  WHEN "100" =>
    speed_pwm <= '1';
  WHEN "101" =>
    speed_pwm <= '1';
  WHEN "110' =>
    speed_pwm <= '1';
  WHEN "111" =>
    speed_pwm <= '1';
  WHEN OTHERS => NULL;
  END CASE;
ELSE
  speed_pwm <= '0';
END IF;

-- Speed Setting #4 of 8 => Neutral (1.5 ms pulse) --
ELSIF count_motor >= 1850 AND count_motor < 1856 THEN
  IF speed /="000" THEN
    CASE speed IS
    WHEN "010" =>
      speed_pwm <= '0';
    WHEN "001" =>
      speed_pwm <= '0';
    WHEN "011" =>
      speed_pwm <= '0';
    WHEN "100" =>
      speed_pwm <= '1';
    WHEN "101" =>
      speed_pwm <= '1';
    WHEN "110" =>
      speed_pwm <= '1';
    WHEN "111" =>
      speed_pwm <= '1';
    WHEN OTHERS => NULL;
    END CASE;
  ELSE
    speed_pwm <= '0';
  END IF;

-- Speed Setting #5 of 8 => Eighth Forward (CCW) --
ELSIF count_motor >= 1856 AND count_motor < 1863 THEN
  IF speed /="000" THEN
    CASE speed IS
    WHEN "010" =>
      speed_pwm <= '0';
    WHEN "001" =>
      speed_pwm <= '0';
    WHEN "011" =>
      speed_pwm <= '0';
    WHEN "100" =>
      speed_pwm <= '0';
    WHEN "101" =>
      speed_pwm <= '1';
    WHEN "110" =>
      speed_pwm <= '1';
    WHEN "111" =>
      speed_pwm <= '1';
    WHEN OTHERS => NULL;
    END CASE;
  ELSE
    speed_pwm <= '0';
  END IF;

-- Speed Setting #6 of 8 => Eighth Forward (CW) --
ELSIF count_motor >= 1863 AND count_motor < 1870 THEN
  IF speed /="000" THEN
    CASE speed IS
    WHEN "010" =>
      speed_pwm <= '0';
    WHEN "001" =>
      speed_pwm <= '0';
    WHEN "011" =>
      speed_pwm <= '0';
    WHEN "100" =>
      speed_pwm <= '0';
    WHEN "101" =>
      speed_pwm <= '1';
    WHEN "110" =>
      speed_pwm <= '1';
    WHEN "111" =>
      speed_pwm <= '1';
    WHEN OTHERS => NULL;
    END CASE;
  ELSE
    speed_pwm <= '0';
  END IF;

-- Speed Setting #7 of 8 => Eighth Reverse (CW) --
ELSIF count_motor >= 1870 AND count_motor < 1877 THEN
  IF speed /="000" THEN
    CASE speed IS
    WHEN "010" =>
      speed_pwm <= '0';
    WHEN "001" =>
      speed_pwm <= '0';
    WHEN "011" =>
      speed_pwm <= '0';
    WHEN "100" =>
      speed_pwm <= '0';
    WHEN "101" =>
      speed_pwm <= '1';
    WHEN "110" =>
      speed_pwm <= '1';
    WHEN "111" =>
      speed_pwm <= '1';
    WHEN OTHERS => NULL;
    END CASE;
  ELSE
    speed_pwm <= '0';
  END IF;

-- Speed Setting #8 of 8 => Eighth Reverse (CCW) --
ELSIF count_motor >= 1877 AND count_motor < 1884 THEN
  IF speed /="000" THEN
    CASE speed IS
    WHEN "010" =>
      speed_pwm <= '0';
    WHEN "001" =>
      speed_pwm <= '0';
    WHEN "011" =>
      speed_pwm <= '0';
    WHEN "100" =>
      speed_pwm <= '0';
    WHEN "101" =>
      speed_pwm <= '1';
    WHEN "110" =>
      speed_pwm <= '1';
    WHEN "111" =>
      speed_pwm <= '1';
    WHEN OTHERS => NULL;
    END CASE;
  ELSE
    speed_pwm <= '0';
  END IF;
-- Speed Setting #6 of 8 => Quarter Forward (CCW) --
ELSIF count_motor >= 1863 AND count_motor < 1875 THEN
  IF speed /="000" THEN
    CASE speed IS
      WHEN "010" =>
        speed_pwm <= '0';
      WHEN "001" =>
        speed_pwm <= '0';
      WHEN "011" =>
        speed_pwm <= '0';
      WHEN "100" =>
        speed_pwm <= '0';
      WHEN "101" =>
        speed_pwm <= '0';
      WHEN "110" =>
        speed_pwm <= '0';
      WHEN "111" =>
        speed_pwm <= '1';
      WHEN OTHERS => NULL;
    END CASE;
    ELSE
      speed_pwm <= '0';
    END IF;
  ELSE
    speed_pwm <= '0';
  END IF;
END PROCESS;

-- Speed Setting #7 of 8 => Half Forward (CCW) --
ELSIF count_motor >= 1875 AND count_motor < 1900 THEN
  IF speed /="000" THEN
    CASE speed IS
      WHEN "010" =>
        speed_pwm <= '0';
      WHEN "001" =>
        speed_pwm <= '0';
      WHEN "011" =>
        speed_pwm <= '0';
      WHEN "100" =>
        speed_pwm <= '0';
      WHEN "101" =>
        speed_pwm <= '0';
      WHEN "110" =>
        speed_pwm <= '0';
      WHEN "111" =>
        speed_pwm <= '1';
      WHEN OTHERS => NULL;
    END CASE;
  ELSE
    speed_pwm <= '0';
  END IF;
END PROCESS;

-- Speed Setting #8 of 8 => Full Forward (CCW) --
ELSIF count_motor >= 1900 AND count_motor < 1925 THEN
  IF speed /="000" THEN
    CASE speed IS
      WHEN "010" =>
        speed_pwm <= '0';
      WHEN "001" =>
        speed_pwm <= '0';
      WHEN "011" =>
        speed_pwm <= '0';
      WHEN "100" =>
        speed_pwm <= '0';
      WHEN "101" =>
        speed_pwm <= '0';
      WHEN "110" =>
        speed_pwm <= '0';
      WHEN "111" =>
        speed_pwm <= '1';
      WHEN OTHERS => NULL;
    END CASE;
  ELSE
    speed_pwm <= '0';
  END IF;
END PROCESS;

-- All of the case statements with <=1 and reoccurring <=0 could be deleted. --
-- PWM for Servo Direction Control

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY direction_control IS
PORT (clock_100kHz : IN STD_LOGIC;
position : IN STD_LOGIC_VECTOR ( 2 DOWNTO 0 );
direction_pwm : OUT STD_LOGIC);
END direction_control ;

ARCHITECTURE servo_controller OF direction_control IS
SIGNAL count_motor: STD_LOGIC_VECTOR ( 10 DOWNTO 0 );
BEGIN
PROCESS
BEGIN
-- Count_motor is a 20ms timer => [ 0 , 2000 ] --
WAIT UNTIL clock_100kHz'EVENT AND clock_100kHz = '1';
IF count_motor /= 2000 THEN
count_motor <= count_motor + 1;
ELSE
count_motor <= "00000000000";
END IF;

-- Direction Setting #1 of 8 => SIGNAL OFF = 000 --
IF count_motor >= 1700 AND count_motor < 1818 THEN
IF position = "000" THEN
direction_pwm <= '0';
ELSE
direction_pwm <= '1';
END IF;

-- At this point the pulse is already 1 ms long --
-- Decide how long to allow pulse to remain high. [ 1ms , 2ms ] --

-- Direction Setting #2 of 8 => 45 degrees LEFT or CW from center (1830) --
ELSIF count_motor >= 1818 AND count_motor < 1830 THEN
IF position /="000" THEN
CASE position IS
WHEN "011" =>

direction_pwm <= '0';
WHEN "010" =>

direction_pwm <= '1';
WHEN "001" =>

direction_pwm <= '1';
WHEN "100" =>

direction_pwm <= '1';
WHEN "101" =>

direction_pwm <= '1';
WHEN "110" =>

direction_pwm <= '1';
WHEN "111" =>

direction_pwm <= '1';
WHEN OTHERS => NULL;
END CASE;
ELSE

direction_pwm <= '0';
END IF;

-- Direction Setting #3 of 8 => 20 degrees LEFT or CW from center (1845) --
-- After trial and error the usable value was determined to be 1830 --
ELSIF count_motor >= 1830 AND count_motor < 1838 THEN
  IF position /="000" THEN
    CASE position IS
      WHEN "011" =>
        direction_pwm <= '0';
      WHEN "010" =>
        direction_pwm <= '0';
      WHEN "001" =>
        direction_pwm <= '1';
      WHEN "100" =>
        direction_pwm <= '1';
      WHEN "101" =>
        direction_pwm <= '1';
      WHEN "110" =>
        direction_pwm <= '1';
      WHEN "111" =>
        direction_pwm <= '1';
      WHEN OTHERS => NULL;
      END CASE;
    ELSE
      direction_pwm <= '0';
    END IF;
  END IF;

-- Direction Setting #4 of 8 => 10 degrees LEFT or CW from center (1850) --
-- After trial and error the usable value was determined to be 1838 --
ELSIF count_motor >= 1838 AND count_motor < 1845 THEN
  IF position /="000" THEN
    CASE position IS
      WHEN "011" =>
        direction_pwm <= '0';
      WHEN "010" =>
        direction_pwm <= '0';
      WHEN "001" =>
        direction_pwm <= '0';
      WHEN "100" =>
        direction_pwm <= '1';
      WHEN "101" =>
        direction_pwm <= '1';
      WHEN "110" =>
        direction_pwm <= '1';
      WHEN "111" =>
        direction_pwm <= '1';
      WHEN OTHERS => NULL;
      END CASE;
    ELSE
      direction_pwm <= '0';
    END IF;
  END IF;

-- Direction Setting #5 of 8 => 0 degrees or center (1856) --
-- After trial and error the usable value was determined to be 1845 --
ELSIF count_motor >= 1845 AND count_motor < 1862 THEN
  IF position /="000" THEN
    CASE position IS
      WHEN "011" =>
        direction_pwm <= '0';
      WHEN "010" =>
        direction_pwm <= '0';
      WHEN "001" =>
        direction_pwm <= '0';
      WHEN "100" =>
        direction_pwm <= '0';
      WHEN "101" =>
        direction_pwm <= '1';
      WHEN "110" =>
        direction_pwm <= '1';
      WHEN "111" =>
        direction_pwm <= '1';
      WHEN OTHERS => NULL;
      END CASE;
    ELSE
      direction_pwm <= '0';
    END IF;
  END IF;

-- Direction Setting #6 of 8 => 10 degrees RIGHT or CCW from center (1870) --
-- After trial and error the usable value was determined to be 1845 --
ELSIF count_motor >= 1862 AND count_motor < 1879 THEN
  IF position /="000" THEN
    CASE position IS
      WHEN "011" =>
        direction_pwm <= '0';
      WHEN "010" =>
        direction_pwm <= '0';
      WHEN "001" =>
        direction_pwm <= '0';
      WHEN "100" =>
        direction_pwm <= '0';
      WHEN "101" =>
        direction_pwm <= '1';
      WHEN "110" =>
        direction_pwm <= '1';
      WHEN "111" =>
        direction_pwm <= '1';
      WHEN OTHERS => NULL;
      END CASE;
    ELSE
      direction_pwm <= '0';
    END IF;
  END IF;

-- Direction Setting #7 of 8 => 0 degrees or center (1880) --
-- After trial and error the usable value was determined to be 1879 --
ELSIF count_motor >= 1879 AND count_motor < 1896 THEN
  IF position /="000" THEN
    CASE position IS
      WHEN "011" =>
        direction_pwm <= '0';
      WHEN "010" =>
        direction_pwm <= '0';
      WHEN "001" =>
        direction_pwm <= '0';
      WHEN "100" =>
        direction_pwm <= '0';
      WHEN "101" =>
        direction_pwm <= '1';
      WHEN "110" =>
        direction_pwm <= '1';
      WHEN "111" =>
        direction_pwm <= '1';
      WHEN OTHERS => NULL;
      END CASE;
    ELSE
      direction_pwm <= '0';
    END IF;
  END IF;

-- Direction Setting #8 of 8 => 10 degrees RIGHT or CCW from center (1888) --
-- After trial and error the usable value was determined to be 1896 --
ELSIF count_motor >= 1896 AND count_motor < 1913 THEN
  IF position /="000" THEN
    CASE position IS
      WHEN "011" =>
        direction_pwm <= '0';
      WHEN "010" =>
        direction_pwm <= '0';
      WHEN "001" =>
        direction_pwm <= '0';
      WHEN "100" =>
        direction_pwm <= '0';
      WHEN "101" =>
        direction_pwm <= '1';
      WHEN "110" =>
        direction_pwm <= '1';
      WHEN "111" =>
        direction_pwm <= '1';
      WHEN OTHERS => NULL;
      END CASE;
    ELSE
      direction_pwm <= '0';
    END IF;
  END IF;
WHEN "111" =>
  direction_pwm <= '1';
WHEN OTHERS => NULL;
END CASE;
ELSE
  direction_pwm <= '0';
END IF;

-- Direction Setting #6 of 8 => 10 degrees RIGHT or CCW from center (1862) --
ELSIF count_motor >= 1862 AND count_motor < 1868 THEN
  IF position /= "000" THEN
    CASE position IS
      WHEN "011" =>
        direction_pwm <= '0';
      WHEN "010" =>
        direction_pwm <= '0';
      WHEN "001" =>
        direction_pwm <= '0';
      WHEN "100" =>
        direction_pwm <= '0';
      WHEN "101" =>
        direction_pwm <= '0';
      WHEN "110" =>
        direction_pwm <= '0';
      WHEN "111" =>
        direction_pwm <= '1';
      WHEN OTHERS => NULL;
    END CASE;
  ELSE
    direction_pwm <= '0';
  END IF;

-- Direction Setting #7 of 8 => 20 degrees CCW from center (1868) --
ELSIF count_motor >= 1868 AND count_motor < 1882 THEN
  IF position /= "000" THEN
    CASE position IS
      WHEN "011" =>
        direction_pwm <= '0';
      WHEN "010" =>
        direction_pwm <= '0';
      WHEN "001" =>
        direction_pwm <= '0';
      WHEN "100" =>
        direction_pwm <= '0';
      WHEN "101" =>
        direction_pwm <= '0';
      WHEN "110" =>
        direction_pwm <= '0';
      WHEN "111" =>
        direction_pwm <= '1';
      WHEN OTHERS => NULL;
    END CASE;
  ELSE
    direction_pwm <= '0';
  END IF;

ELSE
  direction_pwm <= '0';
END IF;

-- Direction Setting #8 of 8 => 45 degrees CCW from center (1882) --
ELSE
  direction_pwm <= '0';
END IF;

END PROCESS;
END servo_controller;

--All of the case statements with <=1 and reoccurring <=0 could be deleted.--
#include "nios.h"
#include "vicon.h"

/*----------------------------------------------------------------- *
* Author(s): Kwabena Asare Bosompem
* Andre Moore
* Jeff Vickers
* Created: May 30th, 2002.
* Description: Functions for Servo Control
* Comments:
*--------------------------------------------------------------*/

/*-----------------------------------------------------------------*
* File Wide Variables *
*--------------------------------------------------------------*
int viconspeed;
int vicondirection;
extern np_pio *direction;
extern np_pio *speed ;

/*-----------------------------------------------------------------*
* Modifiers *
*--------------------------------------------------------------*

/*-----------------------------------------------------------------*
* Accessors *
*--------------------------------------------------------------*/

/*-----------------------------------------------------------------*
* Modifiers *
*--------------------------------------------------------------*

/*-----------------------------------------------------------------*
* File Wide Variables *
*--------------------------------------------------------------*

/*-----------------------------------------------------------------*
* Accessors *
*--------------------------------------------------------------*/

/*-----------------------------------------------------------------*
* Set the servo's speed based on integer passed in
*--------------------------------------------------------------*

int GetSpeed(void)
{
    return viconspeed;
}

int GetDirection(void)
{
    return vicondirection;
}

void SetSpeed(int spd)
if (spd > 7)
    spd = 7;
else if (spd < 0)
    spd = 0;

viconspeed = spd;
speed = (np_pio *) 0x430;
speed->np_piodata = viconspeed;

}//end of SetSpeed

/**
 * Sets the servo's direction based on the integer passed in
 * Created: June 6th, 2002
 * Last Modified: July 15th, 2002
 * Comments: Acceptable range 0-7
 *
 */

void SetDirection(int drx)
{
    if (drx > 7)
        drx = 7;
    else if (drx < 0)
        drx = 0;
    vicondirection = drx;
    direction = (np_pio *) 0x490;
    direction->np_piodata = vicondirection;

}//end of SetDirection
Appendix D:

State Machine C Code & Track
Figure D-1. ViCoN-Bot negotiating a left hand turn.

Figure D-2. Part of ViCoN testing course.
#include "nios.h"
#include "vicon.h"
#include "pio_lcd16207.h"

/*-----------------------------------------------------------------
 * Author(s): Kwabena Asare Bosompem
 * Andre Moore
 * Jeff Vickers
 * Kevin Walker
 * Created: June 10th, 2002
 * Modified: July 18th, 2002
 * Description: Controls the different states needed to use the
 * camera as tracking utility for the robot
 *-----------------------------------------------------------------*/

char state;
int mass = 8; // Value to be used to set the Middle Mass
int count = 0;
int idle_count = 0; // Global used in camera idle
Color color_k;
Color color_checkup;
int packets = 2; // Number of packets used in Tracking()
np_pio *direction = na_direction_pio;
np_pio *speed = na_speed_pio;
int TALK = 0; // 1 is for DEBUG mode

int direction0 = 0;//Stop or Brake?
int direction1 = 1;//-45
int direction2 = 2;//-20
int direction3 = 3;//-10
int direction4 = 4;//0
int direction5 = 5;//10
int direction6 = 6;//20
int direction7 = 7;//45

int speed0 = 0; //Reverse 1/8
int speed1 = 1; //Reverse 1/4
int speed2 = 2; //Stop
int speed3 = 3; //Neutral
int speed4 = 4; //Forward 1/8
int speed5 = 5; //Forward 1/4
int speed6 = 6; //Forward 1/2
int speed7 = 7; //Forward 1

extern int pollmode;
extern SdataPacket spacket;
extern CdataPacket cpacket;
extern MdataPacket mpacket;
extern NdataPacket npacket;

typedef struct {
  int TurnDirection; // number that will hold the direction of the servo
  int DriveSpeed; // number that will hold the speed of the servo
} ServoValue;

ServoValue myServoValue;

**
MAIN ( VOID )
int main(void)
{
    np_pio *pio = na_button_pio; //choose the sw buttons as input
    //set up interrupt handler. this responds to the pushbuttons
    nr_installuserisr(na_button_pio_irq, MyPIO_ISR, (int)pio);
    nr_pio_lcdinit(na_lcd_pio); // Initialize LCD
    pio->np_pioedgecapture = 0;  // clear any existing IRQ
    pio->np_piodirection = 0;    // all input
    pio->np_piointerruptmask = 0xff;  // they all generate irq's
    nr_pio_lcdwritescreen("GaTech ViCoN-Bot"); //Welcome Message
    nr_delay(2000);
    CameraStart();
}

int CameraStart(void)
{
    int ans1;
    int ans2;
    int ans3;
    char tempc[32];
    nr_pio_lcdwritescreen("Camera Initialized");
    nr_delay(1000);
    do{
        ResetCamera();
        ans2 = SetPollMode(1);
        ans1 = SetMiddleMass(mass);
        ans3 = SetColorMode(44);
        }while (ans1 == FAILURE);
        sprintf(tempc, "%d %d %d", ans1, ans2, ans3);
    nr_pio_lcdwritescreen(tempc);
        nr_delay(1000);
        SetSpeed(NEUTRAL); // Set the motor to neutral
    SetDirection(STRAIGHT); // Set the wheels straight
    color_checkup = PINK;
    FindColor();
}
int FindColor(void)
{
    char nicon[32];
    char kevin[32];
    int count2 = 0;
    int conf[3] = {0, 0, 0};
    int conf_value;

    if (count != 2)
    {
        count++;
    }
    else
    {
        count = 0;
    }
    SetSpeed(NEUTRAL);
    SetDirection(STRAIGHT);

    // Add an additional case for another color
    switch (count)
    {
        case(1):
        {
            //color_k = PINK;
            color_k = GREEN;
            if (TALK == 1) {
                nr_pio_lcdwritescreen("Color is now GREEN");
                nr_delay(1100);
            }
            break;
        }
        case(2):
        {
            //color_k = YELLOW;
            color_k = PINK;
            if (TALK == 1) {
                nr_pio_lcdwritescreen("Color is now PINK");
                nr_delay(1100);
            }
            break;
        }
        case(0):
        {
            color_k = PINK;
            if (TALK == 1){
                nr_pio_lcdwritescreen("Color is now YELLOW");
                nr_delay(1100);
            }
            break;
        }
        default:
        {
            break;
        }
    }

    // This do loop controls how many packets are need to find if a color is present
    // Testing has shown that only one packet is needed and exhibits the best functionality
    //do{
    
}
TrackColor(color_k);

// conf[count2] = npacket.confidence; Use if more than one packet is used

if (TALK == 1) {
    sprintf(kevin, "Confidence: %d Pixels:%d", npacket.confidence, npacket.pixels);
nr_pio_lcdwritescreen(kevin);
nr_delay(1000);
}

// Use if tracking more than one packet
if (count2 == 2) {
    conf_value = ((conf[0] + conf[1] + conf[2]) / 3);
    if (TALK == 1) {
        sprintf(kevin, "%d %d %d", conf[0], conf[1], conf[2]);
nr_pio_lcdwritescreen(kevin);
nr_delay(800);
    }
    count2 = count2 + 1;
} else {
    count2 = count2 + 1;
}

} while(count2 < 3);

count2=0;
conf_value=npacket.confidence;

if (TALK == 1) {
    sprintf(kevin, "average confidence: %d", conf_value);
nr_pio_lcdwritescreen(kevin);
nr_delay(1000);
}

if (conf_value > 20) {
    if (TALK == 1) {
        nr_pio_lcdwritescreen("Tracking State");
nr_delay(500);
        TrackState();
    }
} else {
    if (TALK == 1) {
        nr_pio_lcdwritescreen("Finding Color");
nr_delay(500);
        FindColor();
    }
}

/**
 * TRACKING STATE (INT)
 * Description: Tracks a color
 * Created: June 15th, 2002.
 * Parameters:
 * Returns:
 * Comments:
 * Modified:
int TrackState(void){
    char kevin[32];
    int countk = 0;
    int checkup = 0;
    int state_conf = 0;
    int state_mx = 0;
    int state_my = 0;
    int state_x1 = 0;
    int state_y1 = 0;
    int state_x2 = 0;
    int state_y2 = 0;
    int state_pixel = 0;
    int state_confidence = 0;
    if (TALK == 1) {
        nr_pio_lcdwritescreen("Tracking");
        nr_delay(500);
    }
    if (CompareColors(color_k, color_checkup) == SUCCESS){
        TrackColor(color_checkup);
        checkup = npacket.confidence;
        SetDirection(STRAIGHT);
        //SetSpeed(EIGHTH_FORWARD);
        if (checkup <= 20) {FindColor();};
    }
    do{
        TrackColor(color_k);
        state_conf += npacket.confidence;
        state_mx += npacket.mx;
        state_my += npacket.my;
        state_x1 += npacket.x1;
        state_x2 += npacket.x2;
        state_y1 += npacket.y1;
        state_y2 += npacket.y2;
        state_pixel += npacket.pixels;
        countk++;
    }while(countk < packets);
    state_conf = (state_conf/packets);
    state_mx = (state_mx/packets);
    state_my = (state_my/packets);
    state_pixel = (state_pixel/packets);
    if (TALK == 1) {
        sprintf(kevin, "T:..: conf:%d pix:%d", state_conf, state_pixel);
        nr_pio_lcdwritescreen(kevin);
        nr_delay(800);}
    if (state_conf > 20) {DataControl(state_conf, state_mx, state_my, state_pixel);} else {FindColor();}
}/**
* COM P A R E C O L O R S ( S T R U C T , S T R U C T )
* Description: Compares two colors
* Created: June 15th, 2002.
* Parameters: Two structures (Color)
* Returns: int (SUCCESS or FAILURE)
```c
int CompareColors(Color c1, Color c2)
{
    if(c1.rmin != c2.rmin)
        return FAILURE;
    else if (c1.bmin != c2.bmin)
        return FAILURE;
    else if (c1.gmin != c2.gmin)
        return FAILURE;
    else if (c1.rmax != c2.rmax)
        return FAILURE;
    else if (c1.gmax != c2.gmax)
        return FAILURE;
    else if (c1.bmax != c2.bmax)
        return FAILURE;
    else
        return SUCCESS;
}

int DataControl(int con, int m1, int m2, int pix){
    char kevinw[32];

    if (CompareColors(color_k, PINK)== SUCCESS){
        myServoValue.TurnDirection = SetViconDirection(m1);
        if (pix >= 180)
            myServoValue.DriveSpeed = EIGHTH_FORWARD;
        else if ((pix < 180) && (pix >= 25))
            myServoValue.DriveSpeed = EIGHTH_FORWARD;
        else
            myServoValue.DriveSpeed = EIGHTH_FORWARD;
    }
    else if (CompareColors(color_k, GREEN)== SUCCESS){
        if (pix >= 200){
            myServoValue.DriveSpeed = EIGHTH_REVERSE;
            myServoValue.TurnDirection = SetViconDirection();}
        else if ((pix < 200) && (pix >= 125)){
            myServoValue.DriveSpeed = EIGHTH_FORWARD;
            myServoValue.TurnDirection = LEFT_45;}
        else if ((pix < 125) && (pix >= 25)){
            myServoValue.DriveSpeed = EIGHTH_FORWARD;
            myServoValue.TurnDirection = LEFT_20;}
        else
            myServoValue.DriveSpeed = EIGHTH_FORWARD;
```
myServoValue.TurnDirection = SetViconDirection(m1);

if (pix >= 80)
    myServoValue.DriveSpeed = NEUTRAL;
else if ((pix < 80) && (pix >= 40))
    myServoValue.DriveSpeed = EIGHTH_FORWARD;
else
    myServoValue.DriveSpeed = EIGHTH_FORWARD;

if (TALK == 1) {
    sprintf(kevinw, "Direction:%d .::. Speed:%d", myServoValue.TurnDirection,
             myServoValue.DriveSpeed);
    nr_pio_lcdwritescreen(kevinw);
    nr_delay(500);
}

SetDirection(myServoValue.TurnDirection);
SetSpeed(myServoValue.DriveSpeed);
TrackState();

int SetViconDirection(int direc_value){

    if (direc_value >= 75)
        return RIGHT_20;
    else if ((direc_value < 75) && (direc_value >= 65))
        return RIGHT_20;
    else if ((direc_value < 70) && (direc_value >= 60))
        return RIGHT_10;
    else if ((direc_value < 60) && (direc_value >= 40))
        return STRAIGHT;
    else if ((direc_value < 40) && (direc_value >= 30))
        return LEFT_10;
    else if ((direc_value < 30) && (direc_value >= 20))
        return LEFT_20;
    else
        return LEFT_45;
}

RESET (VOID)
Description: Reset will reset the camera and send it back to the initial state of CameraStart
/**
 * Created: June 15th, 2002.
 * Parameters:
 * Returns:
 * Comments:
 * Modified:
 */

int Reset(void){
    int startover;
    do{
        startover = ResetCamera();
    }while(startover == FAILURE);
    CameraStart();
}

/**
 * Created: July 17, 2002
 * Parameters:
 * Returns:
 * Comments:
 * Modified:
 */

int IdleRobot(int value){
    idle_count += value;
    if (idle_count == 2)
    {
        idle_count = 0;
        FindColor();
    }else
    {
        SetDirection(STRAIGHT);
        SetSpeed(NEUTRAL);
    }
}

void MyPIO_ISR(int context)
{
    np_pio *pio = (np_pio *)context;
    char s[32];
    int buttons;
    pio->np_pioedgecapture = 0; //reset interrupt
    buttons = pio->np_piodata; //get button pressed
    sprintf(s,"Button Pressed: %1x",buttons & 0xf);
    if(s[19]!='$' )
    {
        nr_pio_lcdwritescreen(s);
        nr_delay(500);
        if (s[19]=='b'){
            nr_pio_lcdwritescreen("Return to CameraStart");
            nr_delay(1500);
            CameraStart();
        }
        else if(s[19]=='7'){
            nr_pio_lcdwritescreen("Start Travelen Man!");
            nr_delay(1500);
            TravelFind();
        }
        else if(s[19]=='d'){
            nr_pio_lcdwritescreen("Return to CameraStart");
            nr_delay(1500);
            CameraStart();
        }
    }
    else if(s[19]=='f')
    {
        nr_pio_lcdwritescreen(s);
        nr_delay(500);
    }
}
nr_pio_lcdwritescreen("Set Robot Idle");
nr_delay(1500);
IdleRobot(1);

else if(s[19]=='e'){ 
nr_pio_lcdwritescreen("Reset");
nr_delay(1500);
  //if(currentmode == 1)
  //  currentmode = 0;
  //else
  //  currentmode = 1;
  //SetPollMode(currentmode);
  Reset();
}

}
Appendix E:

ViCoN-Bot Data & Pictures
Table E-1. ViCoN-Bot Part List.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tamiya 1/12 Scale M1025 Hummer</td>
</tr>
<tr>
<td>1</td>
<td>Futaba MC330CR FET Speed Control</td>
</tr>
<tr>
<td>1</td>
<td>Altera Nios Board (including equipment &amp; software)</td>
</tr>
<tr>
<td>1</td>
<td>CMUcam (including software)</td>
</tr>
<tr>
<td>1</td>
<td>Null Modem (2 male serial connectors)</td>
</tr>
<tr>
<td>2</td>
<td>7.2 volt 2000mAh batteries</td>
</tr>
<tr>
<td>4</td>
<td>1.5 volt batteries &amp; assembly</td>
</tr>
<tr>
<td>1</td>
<td>5 volt ½ amp regulator</td>
</tr>
<tr>
<td>1</td>
<td>100 µF Electrolytic Capacitor</td>
</tr>
<tr>
<td>1</td>
<td>100 pF Polystyrene Capacitor</td>
</tr>
<tr>
<td></td>
<td>Neon Pink &amp; Green Poster Boards</td>
</tr>
<tr>
<td></td>
<td>Mounting Hardware (i.e. L-bracket, screws, tie wraps)</td>
</tr>
</tbody>
</table>

Figure E-1. Nios board and LCD mounted on M1025.
Figure E-2. CMUcam mounted on robot.

Figure E-3. Under the hood and chassis view of robot.
Figure E-4. Frontal picture of robot.

Figure E-5. Top view of robot.