High Contrast Grayscale Object Tracking
Embedded System Design

Senior Project
Electrical Engineering Department

Author:
Scott Donald Neally

Project Advisor:
Fred DePiero

California Polytechnic State University
San Luis Obispo
December 2009

Table of Contents

Section *Page*

List of Figures

List of Tables

Acknowledgements

I'd like to thank and acknowledge my father, Vic Neally, for all his support and valuable suggestions for progressing through the project. On numerous occasions he has helped shape the ideas that have given rise to the final design of this project. I'd also like to thank Gregory Manyak for his lending me his knowledge of the C329 camera and continued help with software debugging issues. Lastly, I'd like to thank my project advisor Dr. Fred DePiero who helped shape the initial structure of this project and give it direction. Without his insight and knowledge of electronics I would never have accomplished this much.

Abstract

The purpose of this project is to prove that a real-time object-tracking camera can be implemented using the Digilent NEXYS board with Xilinx Platform Studio. With great ambition I set to implement the camera with RGB color tracking of bright green objects. This would make segmentation between the object of interest and its background much easier, allowing less stringent requirements on the background color beyond the object. Because of time restraints, the actual implementation remains as 8-bit grayscale, where the object of interest is intended to be white and highly contrasted from its black background. A simple algorithm is used to track the object by computing the centroid of all 'white' pixels in the field of view and adjusting the focal point of the camera accordingly. The rotation and tilt of the camera are modified via pulse-width modulation to two separate miniature servos controlling each axis.

The resulting embedded system performed almost as well as could be expected. Aside from the inherent restrictions on object and background color, the only other limiting feature was the camera processing speed. The camera had a maximum baud rate of 152,000 bps, requiring 315 milliseconds to transmit image data to the NEXYS board. Assuming some overhead costs associated with capturing and processing the image data, it becomes obvious that this project will never approach real-time capabilities with the current camera module. The actual refresh rate found using this embedded system was 620 milliseconds with baud of 115,200 bps. A video of the result can be found at http://www.youtube.com/watch?v=tapdMPeI_0A.

I. Introduction

Ever since taking the CPE embedded systems series at Cal Poly San Luis Obispo, I have become very interested on the applications of the Digilent NEXYS board and embedded systems in general. The final project that my partner and I made for CPE 329 was a joystick controlled pointing platform, capable of moving in continuous 360-degree rotation with 70 degrees of tilt. From that project I learned how to interface peripheral devices to the NEXYS board, manipulate data, and control servos through pulse-width modulation.

I got the idea for my senior project from an online video showing another object tracking embedded system that used a dsPIC for image processing. Originally, I decided to reproduce the object tracking system I saw on the video by purchasing the same development board and microprocessor. However, after spending the first quarter of senior project just trying to get the development environment working, I decided to switch to the NEXYS board.

I purchased a small camera online that produced an analog black and white signal output. My first inclination was to capture the data and convert it via an A/D converter to save it to the NEXYS board. This proved to be quite an endeavor itself so I finally settled with a small uart-based JPEG camera. This camera had the capability to output a range of image sizes, in RAW or JPEG format, and a selection of color types and data sizes. I knew that many others had had success with the uart capabilities of the NEXYS board and saw this as the most efficient use of time.

II. Background

Object tracking is very important capability in the field of computer vision. It is highly desirable in the development of future defense systems and vehicle automation.

There are three key steps in object tracking: the detection of a moving object, segmenting the object from its background, and tracking the object from frame to frame. Some of the abilities of object tracking are:

- To acquire and lock onto targets to determine object location for destruction
- Autonomous travel through guidance and object avoidance systems
- Create automated surveillance to detect suspicious activities
- Allow human-computer interaction, like gesture recognition
- Gather real-time traffic statistics and direct traffic flow

However, many complexities are introduced due to the compression of the 3D world on a 2D image, noise in images, or complex object motion. There are many cases where the object's shape is complex or changing over time. Also, when more than one object is being tracked there is the chance of partial occlusion, or overlaps between two objects of interest. Another challenge is to make the system invariant to changes in illumination. Lastly because many of the applications of object tracking occur in real-time, there are some heavy image storage and processing requirements that must be fulfilled. The many approaches to object tracking differ primarily based on the way they approach some key questions:

- Which object representation is suitable for tracking?
- Which image features should be used?
- How should the motion of the object be modeled?

III. Requirements

There were several decisions to be made in the fields of object representation, feature selection, object detection, noise filtering, and illumination variability. The requirements for this project were selected to be as simple as possible, while still

accomplishing the goal of “real-time” processing. Point detection (centroid location) was found to require the least number of computations to represent the object of interest. The original intent was to use the RGB color spectrum to process the image with the object of interest represented by a bright green hue. However, it proved simpler and more efficient to use 8-bit grayscale feature selection. With one byte of data, one pixel could be represented completely transferred in one uart transmission. Twelve-bit or 16-bit color information would require at least two bytes (two transmissions) to represent the same pixel information. By choosing one byte transmissions instead of two, the process would run twice as fast because image acquisition takes up most of the program runtime.

I found that there were many possibilities available for noise reduction filtering. However, most of these would take processing time away from the real-time aspect of the project. The only “filter” that was used was a requirement that at least 30 pixels needed to be identified as object pixels. This excluded any case where the camera picks up noise alone without an actual object in view. The field of variable illumination was not very thoroughly covered in this project. After a few experiments it was determined that a threshold of a “white” object was to be above a value of 150 on the scale of 0-255. Unfortunately because of these requirements, the object tracking system requires fairly high contrast to work properly.

IV. Design

The Digilent NEXYS Board FPGA with Microblaze processor was used initialize the embedded system and implement the tracking algorithm. A C328R camera was used to capture real-time images and transmit data to the NEXYS board via uart interface. The baud-rate for the uart camera interface was set to 115,200 bps. The camera was mounted on a platform that uses two miniature servos to rotate and tilt the camera into the correct orientation for tracking. The position of each servo is controlled by pulse-width modulation. The software block diagram and software state diagrams are shown below in Figure 1 and Figure 2 respectively.

IP Cores Used from Xilinx Platform Studio:
(2) Local Memory Bus
(2) Timer/Counters
(1) BRAM Block
(1) MicroBlaze
(2) Uartlite
(1) GPIO
(1) OPB

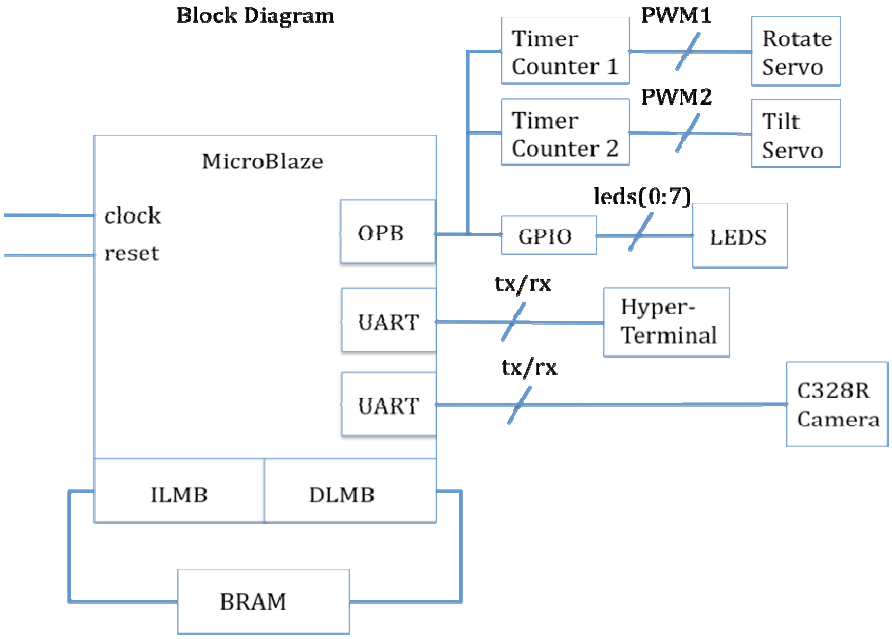

Figure 1 – Software Block Diagram

Initialization:

The timer/counter module for each tilt and rotate servo must be initialized to pulse-width modulation mode. Also, to use the functionality of the LEDs, the GPIO must be set to output mode. Lastly, communication to the camera must be set up and its baud rate established along with the desired parameters for the image size and picture type.

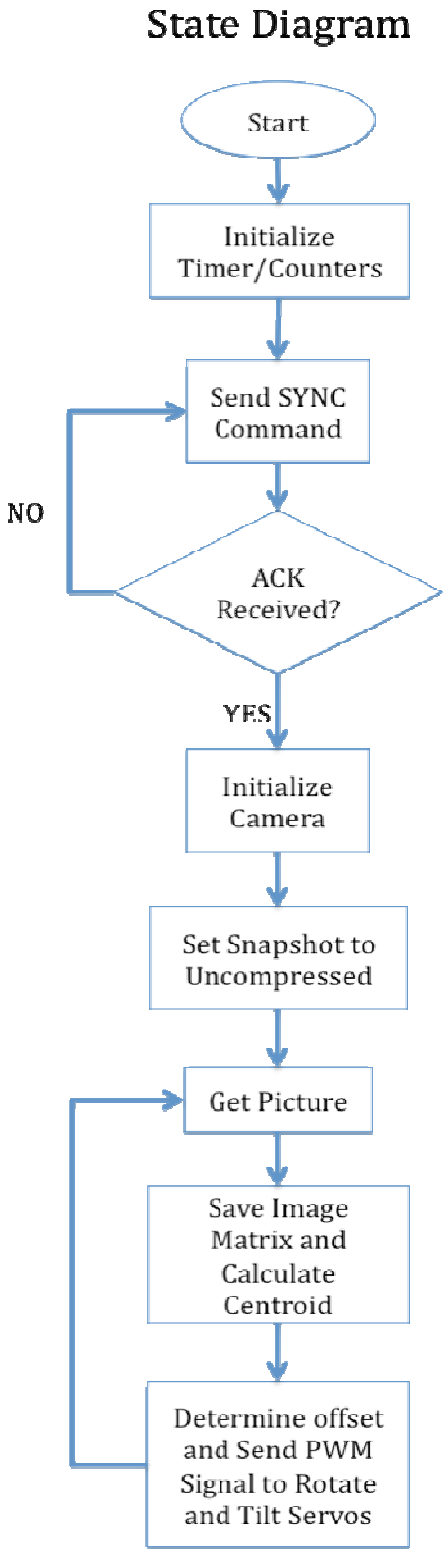

Figure 2 - Software State Diagram

Tilt Servo

- Initialize Load_Register0 to 1000000
- Initialize Load_Register1 to neutral position (i.e. value = 97500)
- Load Timer0 with Load_Register0 Value
- Load Timer1 with Load_Register1 Value
- Initialize Control/Status_Register0 and start timer (i.e. value = 0x696)
- Initialize Control/Status_Register1 and start timer (i.e. value = 0x696)

Rotate Servo

- Initialize Load_Register0 to 1000000
- Initialize Load_Register1 to neutral position (i.e. value = 75000)
- Load Timer0 with Load_Register0 Value
- Load Timer1 with Load_Register1 Value
- Initialize Control/Status_Register0 and start timer (i.e. value = 0x696)
- Initialize Control/Status_Register1 and start timer (i.e. value = 0x696)

LEDS

- Set GPIO to Output Mode
- Blink LED when synchronizing to camera

Camera

- Continuously send SYNC command with 50ms spacing between, until ACK is received
- Initialize camera to 8-bit gray scale, preview resolution 80x60, JPEG resolution 80x64 (i.e. value = 0xAA0100030101)
- Set snapshot parameters to uncompressed picture and skip the first frame before taking picture (i.e. value = 0xAA0501000100)

Object Tracking Algorithm:

Once the camera is initialized, an infinite loop executes that will take a snapshot of the object and surroundings, save the image, segment the object from its background, calculate the centroid of the object, determine the XY adjustment required, and send appropriate PWM signals to the servos to re-center the camera.

Take Picture

- Use get picture command to take a snapshot type picture (i.e. value = 0xAA0401000000)
- Wait, then read ACK command
- Wait, then read image header
- Wait, then read whole image picture
- Send ACK command to camera (i.e. value = 0xAA0E0A00F0F0)

Segment Object from Background and Determine Location

- Determine the centroid of all pixels above some threshold for tracking white images (i.e. the threshold I used was pixel values > 150)

- Assume the center of the camera resides at the center of the 80x64 image; pixel(40,32). Determine the offset of the camera by taking the difference of the centroid and the center pixel.

Update Camera/Servo Location

- Knowing that each servo has a range of motion (pulse widths between 1500us and 2400us, load register count 30,000 and 120,000 respectively) and the fact that tilt servo has cannot use its full range of motion, it has been found that the load register can have a range as follows:
    - Tilt: 40,000 < Load_Register < 110,000
    - Rotate: 30,000 < Load_Register < 120,000
- For the Tilt Servo, determine the direction of the adjustment. For clockwise motion, the increase the current load register value by an increment proportional to the difference between the object centroid and the camera center in the Y direction. Likewise, for counter-clockwise motion, decrease the current load register value proportionally.
- For the Rotate Servo, determine the direction of the adjustment. For clockwise motion, the increase the current load register value by an increment proportional to the difference between the object centroid and the camera center in the X direction. Likewise, for counter-clockwise motion, decrease the current load register value proportionally.

Return to Start

- Reset centroid and counting variables then return to start of algorithm.

V. Development and Construction

I first tested to see that I could connect to the camera and receive data from it at low speeds; I chose 9600 baud for the initial test. The camera has the ability to auto-detect the baud rate of the host device. Therefore I implemented a loop that executed as long as there was no data from the receive FIFO, meaning that it would execute until the camera had sent an acknowledge signal back. I hooked up an oscilloscope probes to the transmit and receive terminals on the camera to view the incoming waveforms shown in **Figure 3**.

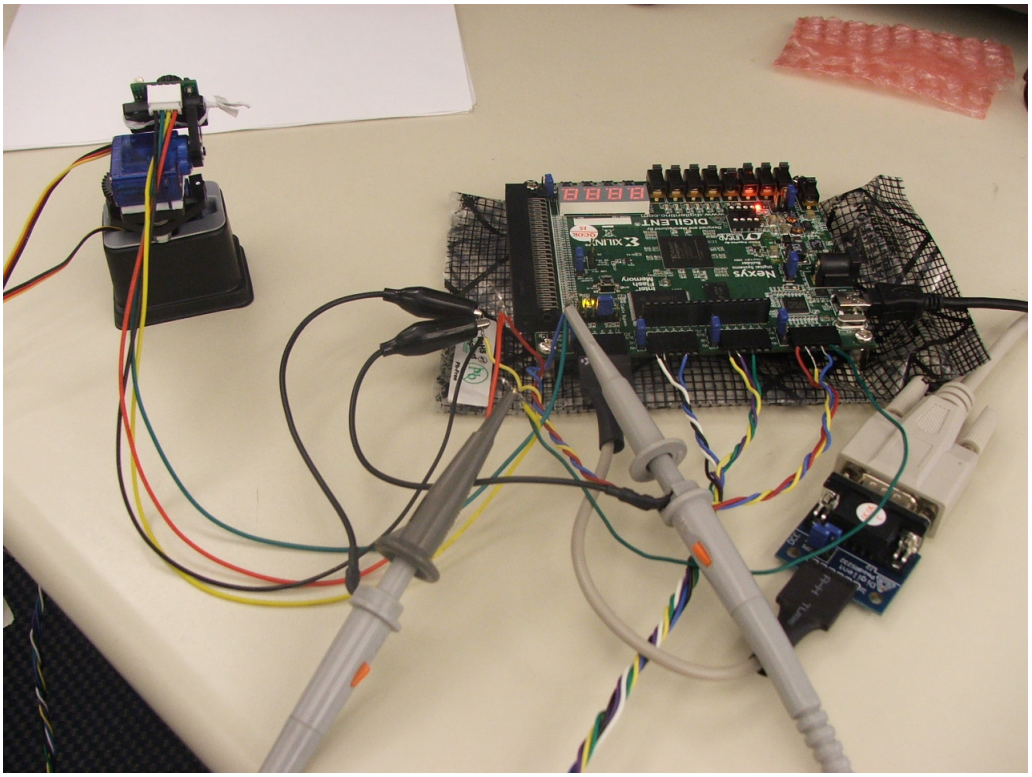

Figure 3 - NEXYS Board with Oscilloscope Probes

The initialization process is shown below in Figure 4. I discovered that the camera required 50ms between SYNC signals to connect correctly. The first three yellow lines in the figure below are the SYNC command being sent to the camera and the first blue line in the ACK command being received.

After delay of 100ms, the first initialization command (last yellow line) is sent to the camera and an ACK command received (last blue line). Once this process was proven at low speeds, I increased the baud rate of the system to 115,200 bps to increase the image transfer rate.

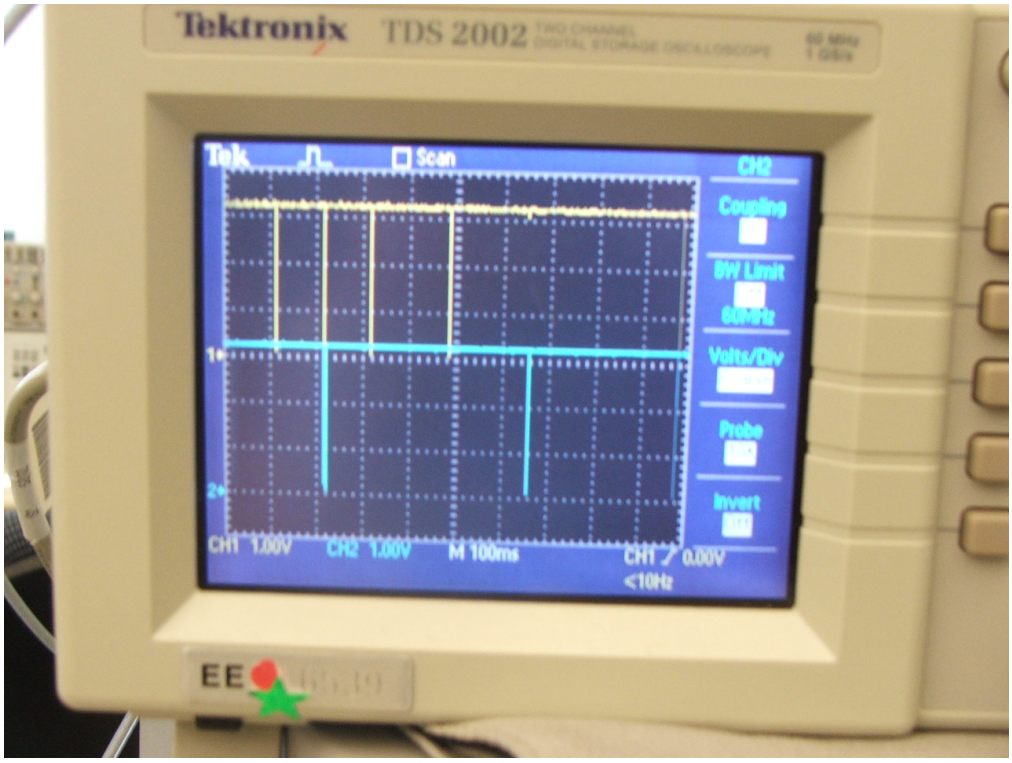

Figure 4 - SYNC Camera

The initialization sequence of the camera sets the image size, color information, number of frames to skip before capturing an image, and image format (JPEG or RAW). After these commands were successfully executed and confirmed by acknowledgements the camera was ready to capture an image. Using the 'get picture' command, the camera captured one 80x60, 8-bit grayscale, RAW format image as shown below in Figure 5. The duration of the image transfer was found to be 420 milliseconds. This agrees with calculated the results:

(60 * 80 pixels + 12 bytes of header data) * 10 bits/pixel / 115200 bps = 417.7 mS

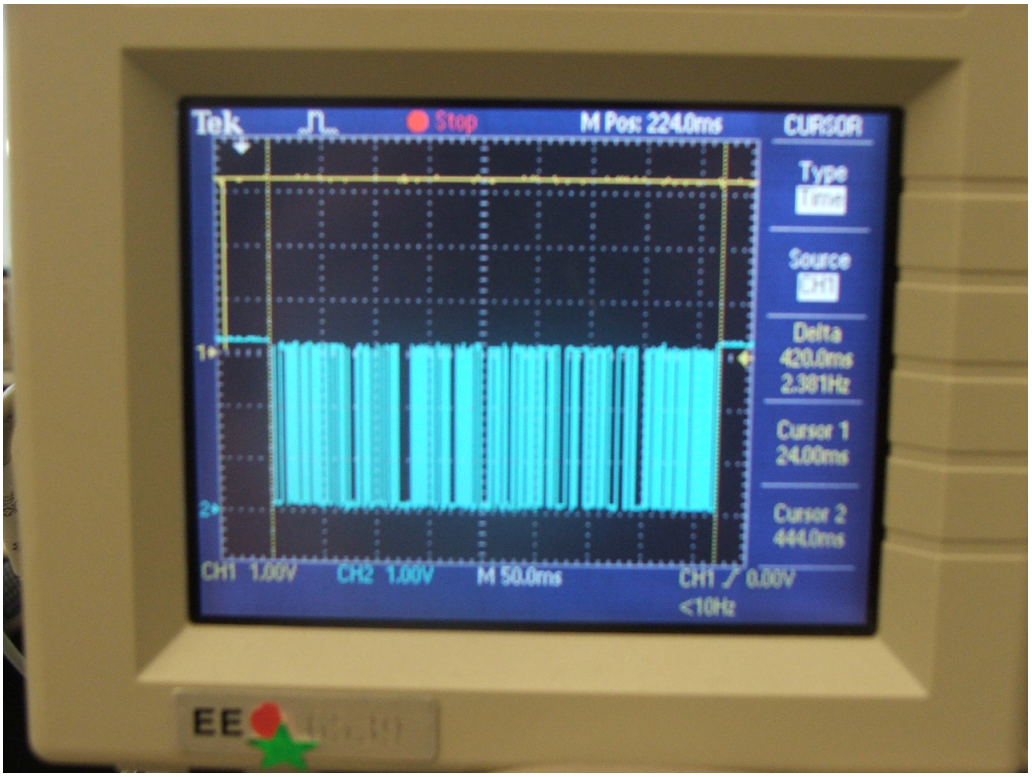

Figure 5 - Camera Image Capture

As the image data was read in, it was filtered to find pixel values greater than 150 to segment the white image pixels from their background. Each pixel's row and column were saved to determine the average value of each, and consequently the combination of the two determined the centroid. The centroid location was subtracted from a chosen center of reference (row 30, column 40) and then multiplied by an experimentally determined gain factor. The gain factor was chosen so that the result would add to the number representing the original servo position and move the camera a moderate amount in the direction of the object.

Once this was successfully implemented, an infinite loop was used to repeat the process indefinitely. Figure 6 below shows the loop running and multiple images being captured while the camera moves to follow the object.

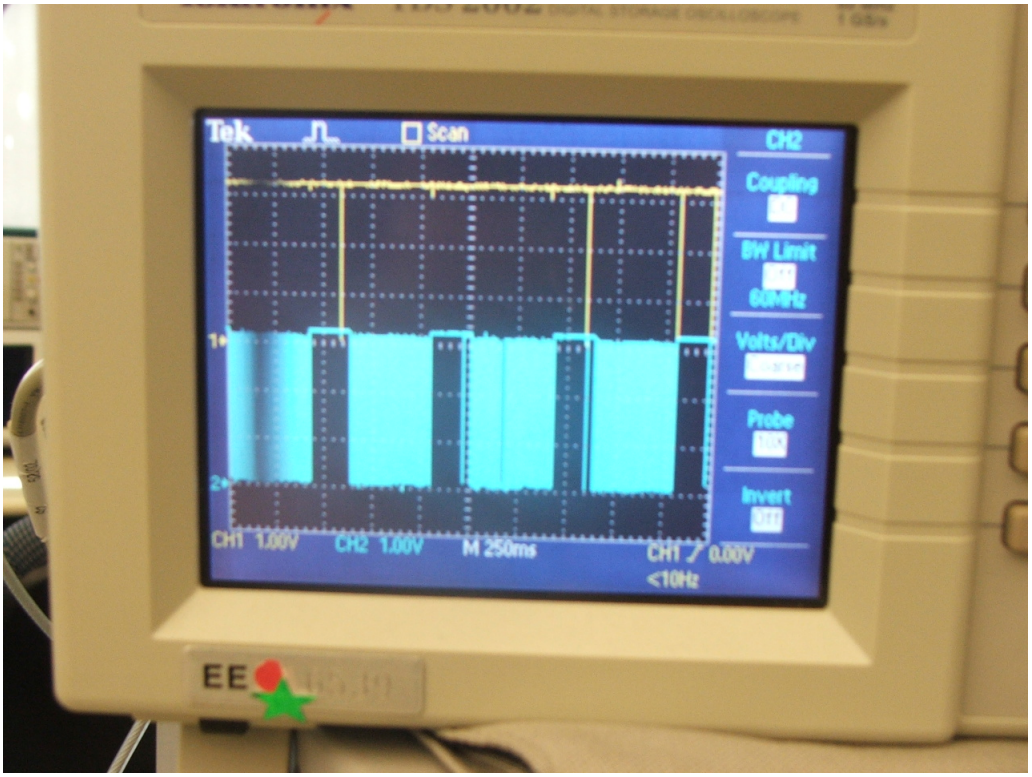

Figure 6 - Multiple Image Captures

From this oscilloscope trace it was found that the average time between image captures was 620 milliseconds, this is the refresh rate of the embedded system. This value is quite slow for real-time applications. A video of the result can be found at http://www.youtube.com/watch?v=tapdMPeI_0A. As is seen in the video, the object must be moved quite slowly to maintain visual tracking.

VI. Integration and Test Results

To determine the cameras ability to discern black and white images, an experiment was conducted to determine the range of pixel grayscale values for pure black and pure white objects. Using an eight-bit grayscale range (0-255) it was found that most black objects spread a range of values between 0 and 50 while white objects inhabited ranges as low as 150 to the maximum of 255. Next to be able to differentiate between an object (ping pong ball) and its background, a small stationary white object (ping pong ball) was introduced on a black background and the output was compared for accuracy and clarity. The result is shown in **Figure 7** below.

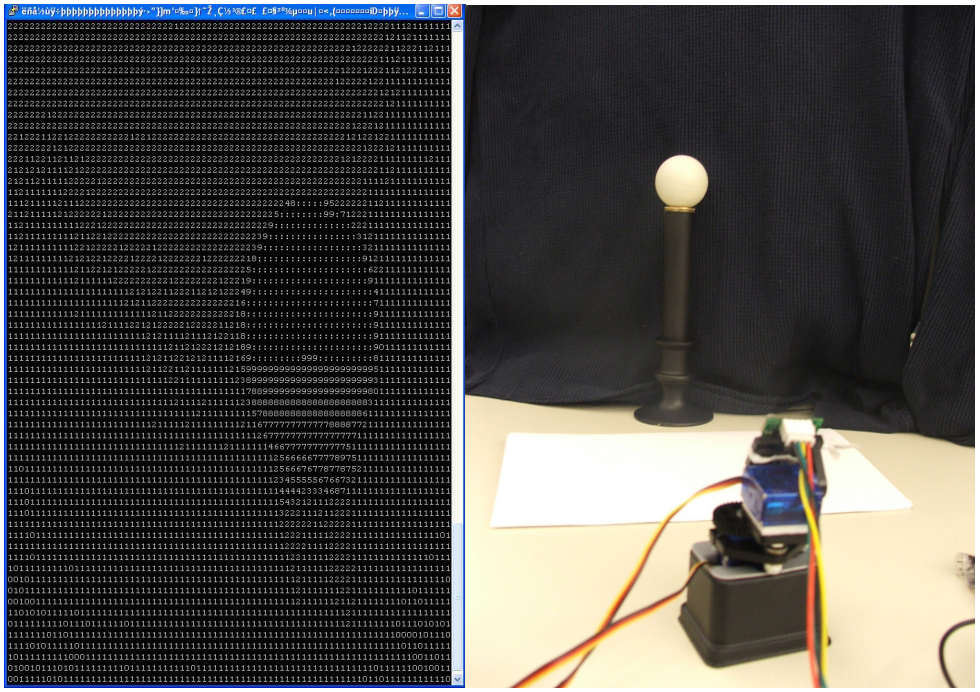

Figure 7 - Camera Initialization on Stable Object

The ASCII image seen on the left hand side uses several 'buckets' to generate the output file in PuTTY. When the image is saved, each pixel location is given a character based on its 8-bit value. For this run of the program I had segmented the 0-255 color space into 11 different sections, the first 10 'buckets' were equally spaced with separations of 25

units each and the 11th bin took all values of color above a value of 250. The character mapping used was ‘0’ through ‘:’ in ASCII.

Next, I moved the ping ping ball through a controlled black background while wearing black gloves. Also, I wanted to obtain a higher contrasting ASCII image, so I searched the internet for a common ASCII character mapping. The website http://www.cs.umd.edu/Outreach/hsContest98/questions/node4.html gave me a good black and white representation that I used instead of the ‘0’ to “:” that had been used before. Let X denote the grayscale value and the character mapping is as follows:

$0 \le X \le 24$ write ‘.’	$25 \le X \le 49$ write ‘,’	$50 \le X \le 74$ write ‘;’	$75 \le X \le 99$ write ‘!’
$100 \le X \le 124$ write ‘v’	$125 \le X \le 149$ write ‘l’	$150 \le X \le 174$ write ‘L’	$175 \le X \le 199$ write ‘F’
$200 \le X \le 224$ write ‘E’	$225 \le X \le 249$ write ‘$’	$250 \le X$ write ‘#’	

The result is show below in **Figure 8**. If you look closely in the picture on the left, you can see the contrast between the black gloves and the background.

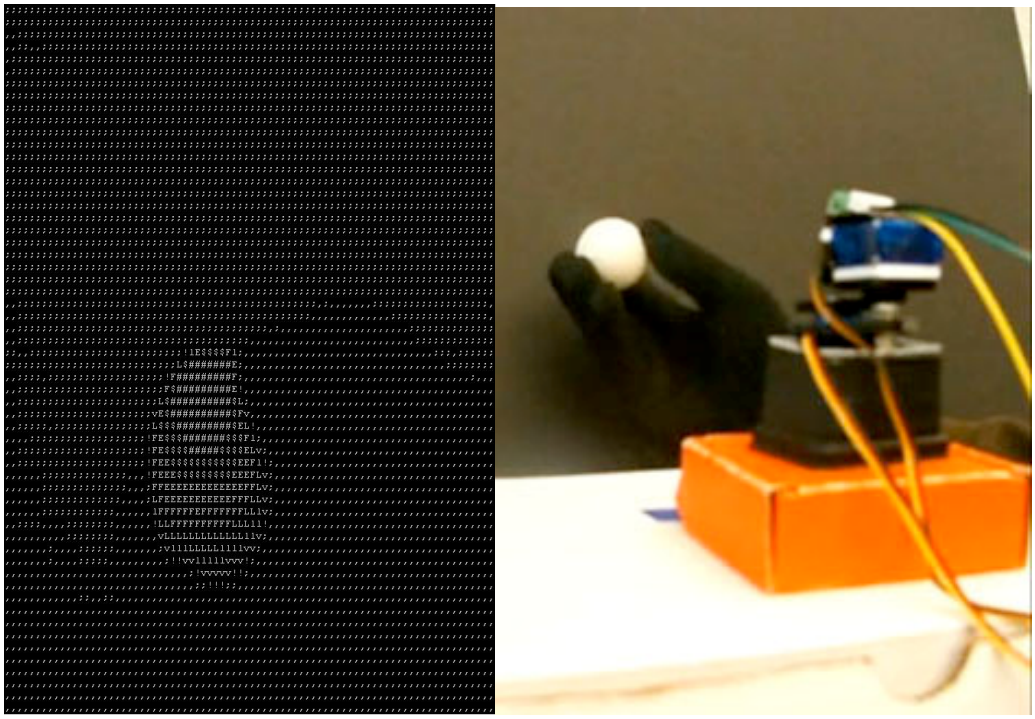

Figure 8 – Image Tracking of Ping-Pong Ball Held by Black Gloves

VII. Conclusion

Overall, I am quite satisfied with the results of this project. There are numerous improvements that could be made to this project, however the proof of concept is quite remarkable and is a good starting point for future development. I have proven that the Digilent NEXY board is capable of nearly real-time image processing and object tracking. The critical restraints that the system faced were based upon the upper limits of the image transfer rate of the uart camera. Minimizing the pixel data width to 4-bit or 2-bit image encoding instead of eight could in turn reduce the image capture time. This would allow more than one pixel data to be received for each uart transfer and reduce the capture time to 210 or 105 mS respectively! As is however, the system still contains an overhead image processing time of 200 mS. This could be reduced by saving the data using an interrupt based system and devote the time remaining in between pixel transfers to dynamically calculate the centroid of all object-classified pixels and decide how much to move the servos.

Another shortcoming of the project was the stringent requirement of the high contrast object and background. As is, the camera requires the background to be dark to black and the object to be colored white. Implementing the system with a 12 or 14-bit RGB color scheme and requiring the object to be of a high intensity green hue would make it easier to segment the object from most backgrounds. This would loosen up the requirements for the background color, however the object color would need to remain fixed. Green hue should be chosen because in 16-bit RGB the green component is given a width of 6 pixels because the human eye has the highest sensitivity for green shades. At this juncture, lighting variance would need to be accounted for and neutralized in some manner.

I found this project to be quite an undertaking. Looking back, I have found that issues with timing constraints created the most difficulty in making progress. Most of the time spent debugging was because of timing issues. I found that often a pause was required in the program to allow a command to finish before processing the next from the camera. I believe this was because the system clock of the Xilinx board was able to run commands much faster (about 2 clock cycles of 50MHz) than commands could be sent or received from the camera (115200 baud). During the image transfer, it became apparent that it was critical to wait until the next byte of data from the camera was received (found by determining if the RX_FIFO contained data) before trying to read it from the stack. Also, it was crucial to read all acknowledgements sent from the camera to ensure the receive FIFO did not overflow.

One infrequent observation that I found was that when the camera first started, the first image frame contained pixels that were overall darker in color (lower value) by twice as much. I found that taking pictures after the 3rd image frame solved this issue. As I progressed throughout the project, I found that the RS232 peripheral was quite an invaluable debugging tool. It allowed me to export data to a program like hyper-terminal (PuTTY was used mostly) to view data or variables in memory. I used PuTTY to view the ASCII image as seen by the camera and segmented into 10 levels. Also, this allowed me to view the contents of different variables as they were saved or computed. In the beginning of the project, the camera would not always accept SYNC commands from the Xilinx board and fail to initialize, sending the Xilinx board into an infinite loop. To observe this state, however, it was required to hook up scope probes to the transmit and receive ports of the camera. In order to observe this case more easily, I set an LED to blink as long as the Xilinx board was trying to SYNC with the camera, this allowed me to

discontinue the use of the scope probes for detection. The source of this problem is yet to be determined.

For almost the entirety of the project, I didn't have any tools to clearly view the image taken by the camera. My fear in this regard was that the camera might not be focused properly. However, once I was able to cleanly save the image data, I was able to view the resulting image using '0' to ':' scaled values. The steps of grayscale were clear, however the overall image was hard to see visually. Therefore I researched ASCII grayscale color mapping. Using the camera with eleven steps of grayscale leveling and character mapping found online, the following self-portrait was produced in **Figure 9**.

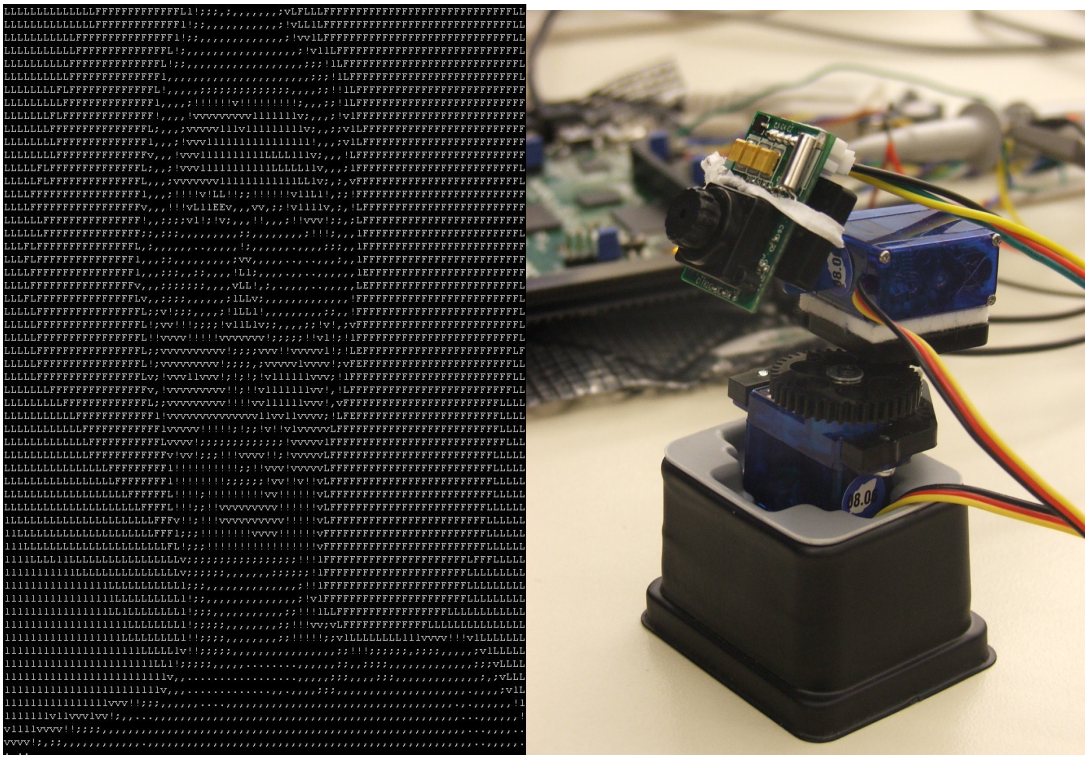

Figure 9 - ASCII Portrait of Myself & Camera Mount on Servo Module

VIII. Bibliography

"ASCII Table and Description." 8 Dec. 2009. <http://www.asciitable.com/>

Tatham, S. “PuTTY: A Free Telnet/SSH Client” Cambridge, England. 24 Jun 2006. 8 Dec. 2009. <http://www.chiark.greenend.org.uk/~sgtatham/putty/>

Tseng, Chau-Wen. “ASCII Art.” 24 Mar. 1998. 8 Dec. 2009. <http://www.cs.umd.edu/Outreach/hsContest98/questions/node4.html>

Yilmaz, A., Javed, O., & Shah M. (2006). Object Tracking: A Survey. ACM Computing Surveys, Volume 38, Issue 4, Article No. 13. New York, NY: ACM

Wikipedia. “Highcolor” 3 Oct. 2009. 8 Dec. 2009. <http://en.wikipedia.org/wiki/Highcolor>

Appendix A. Specifications

Digilent NEXYS Board

Symbol	Parameter	Min	Typ	Max	Units
VDD	DC supply voltage	5		9	V
	Flash ROM			16	Mbytes
	RAM			16	Mbytes
CLK_{sys}	System Clock	25	50	100	MHz

Digilent PMOD-RS232

Symbol	Parameter	Min	Typ	Max	Units
VCC	DC supply voltage	3.0		5.0	V
Logic '0'	Logic Low	+3		+12	V
Logic '1'	Logic High	-3		-12	V

C329R Camera

Symbol	Parameter	Min	Typ	Max	Units
VCC	DC supply voltage	3.0	3.3	3.6	V
Io	Operating Current		60		mA
V_{IH}	High Level Input	2.0			V
V_{IL}	Low Level Input			0.8	V

Hi-Tech HS-55 Sub-Micro Servo

Symbol	Parameter	Min	Typ	Max	Units
VCC	DC supply voltage	4.8	5.0	6.0	V
I_{idle}	Operating Current (idle)		5.4		mA
I_{NL}	Operating Current (no load)		150		mA
V_{pulse}	Range, PWM Pulse Width (-90 to 90 degrees)	600	1500	2400	uSec
V_{pp}	PWM Pulse Peak to Peak	3		5	V

Table 1 - Electrical Specifications

Software Features

- Camera baud rate: 115200 bps — Fastest rate camera handles
- RS232 baud rate: 115200 bps
- Camera Refresh Rate: 620 milliseconds — Found by O-scope
- Image ratio: 80x60 pixels — Smallest image size
- Image size: 9.6 Kb — (60*80*2bytes = 9.6Kb)

Hardware Features

- Blinking light indicates camera is synchronizing to Xilinx board
- Camera elevation from 0 degrees horizontal to 90 degrees vertical
- Camera rotation of -180 to +180 degrees

Appendix B. Parts List and Costs

Part Designation	Manufacturer	Manufacturer PN	Cost
Nexys Xilinx Board	Digilent	NEXYS	$89.00
(2) Miniature Servos	HiTec	HS-55	$27.98
Sub-Micro Pan/Tilt	Servo City	SPT50	$19.99
PMOD RS232	Digilent	PMOD-RS232	$14.99
RS232 Cable (M-F)	Assmann Electronics Inc	AK178-3-R	$7.73
JPEG UART Camera	Spark Fun Electronics	SEN-09334	$54.95
Misc Wiring	N/A	N/A	N/A
TOTAL			$214.64

Table 2 - Bill of Materials

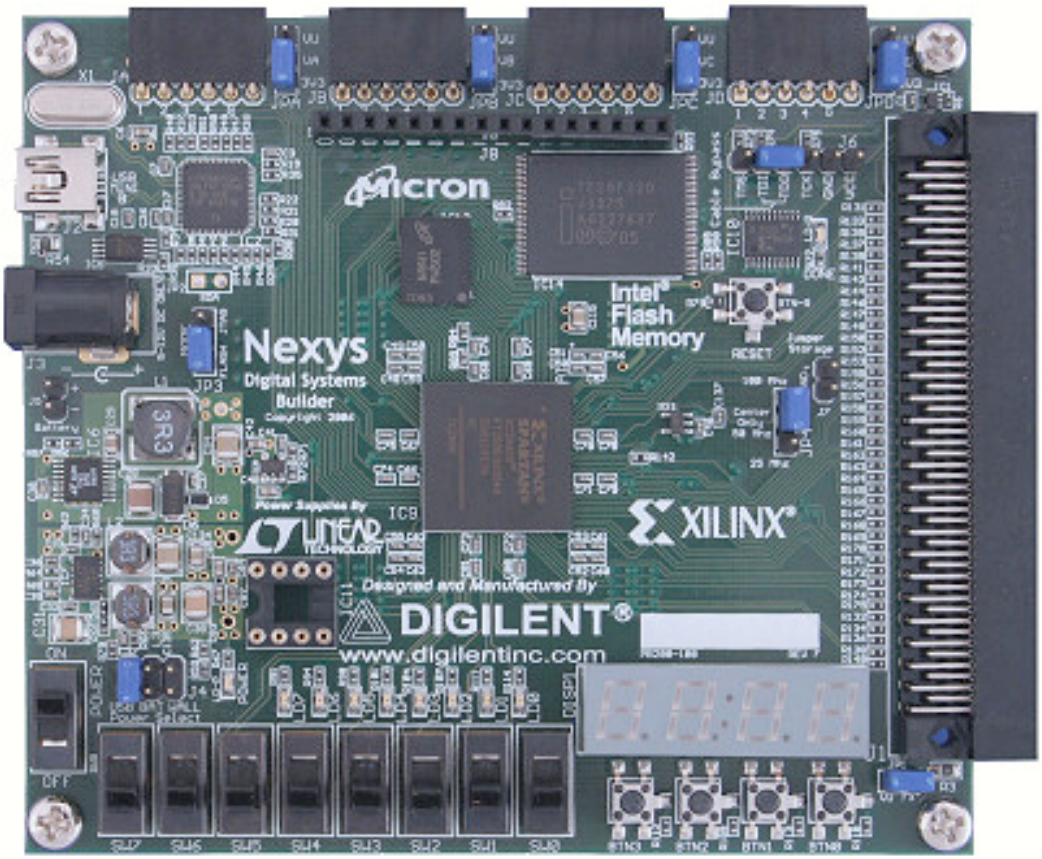

Figure 10 - Digilent NEXYS Board

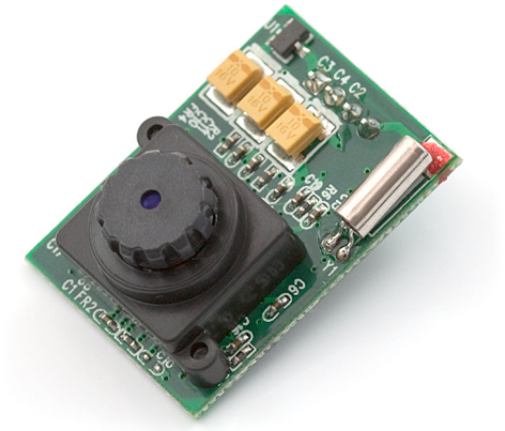

Figure 11 - C329R UART Camera and Digilent PMOD-RS232 Peripheral

Figure 12 - Hi-Tech Sub-Micro Servos with Pan/Tilt Add-on

Appendix C. Hardware Configuration & Wire List

From Component	Designator	FPGA Pin	Signal Type	To Component	Designator
NEXYS 6-pin JA	JA-2	R13	RX	C328R Camera	TxD
NEXYS 6-pin JA	JA-3	T13	TX	C328R Camera	RxD
NEXYS 6-pin JA	JA-5	-	GND	C328R Camera	GND
NEXYS 6-pin JA	JA-6	-	VCC	C328R Camera	VCC
NEXYS 6-pin JB	JB-3	P8	PWM	Rotate Servo	SIG
NEXYS 6-pin JB	JB-5	-	GND	Rotate Servo	GND
NEXYS 6-pin JB	JB-6	-	VCC	Rotate Servo	VCC
NEXYS 6-pin JC	JC-2	P9	PWM	Tilt Servo	SIG
NEXYS 6-pin JC	JC-5	-	GND	Tilt Servo	GND
NEXYS 6-pin JC	JC-6	-	VCC	Tilt Servo	VCC
NEXYS 6-pin JD	JD-3	C10	RX	PMOD RS232	Pin 3
NEXYS 6-pin JD	JD-4	D12	TX	PMOD RS232	Pin 4
NEXYS 6-pin JD	JD-5	-	GND	PMOD RS232	Pin 5
NEXYS 6-pin JD	JD-6	-	VCC	PMOD RS232	Pin 6

Table 3 - Wire List

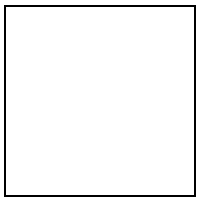

Figure 13 - Hardware Configuration

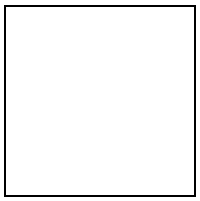

Figure 14 - NEXYS Board Breakout with RS232 Connector

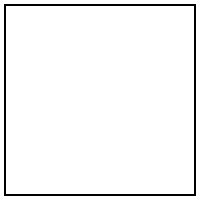

Figure 15 - Close Up of Wiring Breakout
Left to Right: RS232 Connector, Tilt Servo, Rotate Serve, Camera Connector

Appendix D. C Code

```
//Object Tracking Senior Project
//Scott Neally
//Fall 2009

#include "xparameters.h"
#include "xio.h"
typedef int bool;
#define true 1
#define false 0
#define delay_count 136000000                    //count for spin delay of .1 sec
#define setreg (0x696)
#define height 60

int read(){
   while(!(XIo_In32(XPAR_UART_BASEADDR + 8) & 0x01));
   return XIo_In32(XPAR_UART_BASEADDR);
}

void uartTX(char value){
   while(!(XIo_In32(XPAR_UART_BASEADDR + 8) & 0x04));
          XIo_Out32(XPAR_UART_BASEADDR + 4,value);
}

void send_SYNC_cmd(){
          uartTX(0xAA);
          uartTX(0x0D);
          uartTX(0x00);
          uartTX(0x00);
          uartTX(0x00);
          uartTX(0x00);
}

void send_ACK_cmd(){
          uartTX(0xAA);
          uartTX(0x0E);
          uartTX(0x0A);
          uartTX(0x00);
          uartTX(0x00);
          uartTX(0x00);
}

void send_ACK_PIC(){
          uartTX(0xAA);
          uartTX(0x0E);
          uartTX(0x0A);
          uartTX(0x00);
          uartTX(0x00);
          uartTX(0x00);
}

int uartRX(){
          long int value;
          value = read();
   XIo_Out32(XPAR_RS232_BASEADDR + 4,((value>>4) & 0xF) + '0');
   XIo_Out32(XPAR_RS232_BASEADDR + 4,(value & 0xF) + '0');
```

```
    value = value << 8;
            value |= read();
    XIo_Out32(XPAR_RS232_BASEADDR + 4,((value>>4) & 0xF) + '0');
    XIo_Out32(XPAR_RS232_BASEADDR + 4,(value & 0xF) + '0');
            value = value << 8;
            value |= read();
    XIo_Out32(XPAR_RS232_BASEADDR + 4,((value>>4) & 0xF) + '0');
    XIo_Out32(XPAR_RS232_BASEADDR + 4,(value & 0xF) + '0');
            value = value << 8;
            value |= read();
    XIo_Out32(XPAR_RS232_BASEADDR + 4,((value>>4) & 0xF) + '0');
    XIo_Out32(XPAR_RS232_BASEADDR + 4,(value & 0xF) + '0');
            value = value << 8;
            value |= read();
    XIo_Out32(XPAR_RS232_BASEADDR + 4,((value>>4) & 0xF) + '0');
    XIo_Out32(XPAR_RS232_BASEADDR + 4,(value & 0xF) + '0');
            value = value << 8;
            value |= read();
    XIo_Out32(XPAR_RS232_BASEADDR + 4,((value>>4) & 0xF) + '0');
    XIo_Out32(XPAR_RS232_BASEADDR + 4,(value & 0xF) + '0');
    XIo_Out32(XPAR_RS232_BASEADDR + 4,' ');
    XIo_Out32(XPAR_RS232_BASEADDR + 4,' ');
            return value;
}

void RS232TX(char value){
            volatile int status = 0x00;
            //while (!(status && 0x04)) {
            //                      status = XIo_In32(XPAR_UART_BASEADDR + 8);
            //}
            XIo_Out32(XPAR_RS232_BASEADDR + 4,value);
}

volatile short int im[height][80];
short int header[6];
short int ack[6];

main(){
            short int value = 0x00;
            int h,i,j;
            int leds=0;
            bool RX = 0;
            long int ACK;
            long int HEADER;
            long int SYNC;
            int count;
            int col;
            int row;
            int av_X;
            int av_Y;
            int CX = 40;
            int CY = 30;
            int ind;

            //600us, dutycount = 30000; 1500us, dutycount = 75000;2400us, dutycount = 120000
            volatile int duty_count1 = 110000;              //Tilt
            volatile int duty_count2 = 75000;               //Rotate
```

```
        //Initialize PWM Tilt Servo
        XIo_Out32(XPAR_TILT_BASEADDR + 0x04, 1000000);                  //Load Register0
needs to be intialized
        XIo_Out32(XPAR_TILT_BASEADDR + 0x14, duty_count1);              //Load Register1
needs to be intialized
        XIo_Out32(XPAR_TILT_BASEADDR, 0x20);                            //Load Timer0
with Load Register0 Value
        XIo_Out32(XPAR_TILT_BASEADDR + 0x10, 0x20);                     //Load Timer1
with Load Register1 Value
        XIo_Out32(XPAR_TILT_BASEADDR, setreg);                          //Control/Status
Register0 (TCSR0) needs to be intialized and start timer
        XIo_Out32(XPAR_TILT_BASEADDR + 0x10, setreg);                   //Control/Status
Register1 (TCSR1) needs to be intialized and start timer

        //Initialize PWM Rotate Servo
        XIo_Out32(XPAR_ROTATE_BASEADDR + 0x04, 1000000);                //Load Register0
needs to be intialized
        XIo_Out32(XPAR_ROTATE_BASEADDR + 0x14, duty_count2);            //Load Register1
needs to be intialized
        XIo_Out32(XPAR_ROTATE_BASEADDR, 0x20);                          //Load Timer0
with Load Register0 Value
        XIo_Out32(XPAR_ROTATE_BASEADDR + 0x10, 0x20);                   //Load Timer1
with Load Register1 Value
        XIo_Out32(XPAR_ROTATE_BASEADDR, setreg);                        //Control/Status
Register0 (TCSR0) needs to be intialized and start timer
        XIo_Out32(XPAR_ROTATE_BASEADDR + 0x10, setreg);                 //Control/Status
Register1 (TCSR1) needs to be intialized and start timer

XIo_Out32(XPAR_LED_BASEADDR + 4, 0);            // Set GPIO to Output Mode
  for(h=0; h<delay_count; h++);

XIo_Out32(XPAR_RS232_BASEADDR + 4, 0x0A);       //Newline

   while(!RX){                                  //While RX does not contain valid data, send sync cmd
        for(h=0; h<delay_count/8*5; h++);                               //Delay for 100ms
        leds = (leds+1)%2;                                              //Blink While Connecting
        XIo_Out32(XPAR_LED_BASEADDR, leds);                             //Write value to GPIO
        send_SYNC_cmd();                                                //Send SYNC Command
        RX = (XIo_In32(XPAR_UART_BASEADDR + 8) && 0x01);                //Read from Status Reg
}

ACK = uartRX();
for(h=0; h<delay_count; h++);
SYNC = uartRX();
uartTX(0xAA);                                                           //ACK (AA 0E 0D 00 00 00)
uartTX(0x0E);
uartTX(0x0D);
uartTX(0x00);
uartTX(0x00);
uartTX(0x00);

//Write to RS232
for(h=0; h<delay_count; h++);
uartTX(0xAA);                                                           //Initialize (AA 01 00 03 01 01))
uartTX(0x01);
uartTX(0x00);
uartTX(0x03);
uartTX(0x01);
```

```
uartTX(0x01);
for(h=0; h<delay_count; h++);
ACK = uartRX();

for(ind = 0; ind<25;ind++){                                     //Take New Picture Loop
        av_X = 0;
        av_Y = 0;
        row = 0;
        col = 0;
        count = 0;

        for(h=0; h<delay_count; h++);
        uartTX(0xAA);                                           //Snapshot (AA 05 01 00 03 00)
        uartTX(0x05);
        uartTX(0x01);
        uartTX(0x00);
        uartTX(0x03);
        uartTX(0x00);
        ACK = uartRX();
        for(h=0; h<delay_count/4000; h++);
        uartTX(0xAA);                                           //Get Picture AA 04 01 00 00 00)
        uartTX(0x04);
        uartTX(0x01);
        uartTX(0x00);
        uartTX(0x00);
        uartTX(0x00);
        ACK = uartRX();
        HEADER = uartRX();
        for(h=0; h<delay_count/4000; h++);
        XIo_Out32(XPAR_RS232_BASEADDR + 4, 0x0C);                       //New Page

        for (i=0;i<height;i++){
                for (j=0;j<80;j++){
                        while(!(XIo_In32(XPAR_UART_BASEADDR + 8) & 0x01));          //Wait
for valid Data
                        value = XIo_In32(XPAR_UART_BASEADDR);                       //Read 8
bits from Camera
                        if ((value >= 0) && (value <= 24)){                 //ASCII ART MAPPING
                                im[i][j] = '.';
                        }
                        else if ((value >= 25) && (value <= 49)){
                                        im[i][j] = ',';
                        }
                        else if ((value >= 50) && (value <= 74)){
                                        im[i][j] = ';';
                        }
                        else if ((value >= 75) && (value <= 99)){
                                        im[i][j] = '!';
                        }
                        else if ((value >= 100) && (value <= 124)){
                                        im[i][j] = 'v';
                        }
                        else if ((value >= 125) && (value <= 149)){
                                        im[i][j] = 'l';
                        }
                        else if ((value >= 150) && (value <= 174)){
                                        im[i][j] = 'L';
                        }
```

```
                else if ((value >= 175) && (value <= 199)){
                        im[i][j] = 'F';
                }
                else if ((value >= 200) && (value <= 224)){
                        im[i][j] = 'E';
                }
                else if ((value >= 225) && (value <= 249)){
                        im[i][j] = '$';
                }
                else if (value >= 250){
                        im[i][j] = '#';
                }
                if (value >= 150){                              //Threshold for "white" pixels
                        count++;
                        row = row + i;
                        col = col + j;
                }
                XIo_Out32(XPAR_RS232_BASEADDR + 4,im[i][j]);
        }
}

send_ACK_PIC();
av_X = col/count;
av_Y = row/count;

XIo_Out32(XPAR_RS232_BASEADDR + 4,'X');
for(h=0; h<delay_count/4000; h++);
XIo_Out32(XPAR_RS232_BASEADDR + 4,((av_X>>8) & 0xF) + '0');
XIo_Out32(XPAR_RS232_BASEADDR + 4,((av_X>>4) & 0xF) + '0');
XIo_Out32(XPAR_RS232_BASEADDR + 4,(av_X & 0xF) + '0');
for(h=0; h<delay_count/4000; h++);
XIo_Out32(XPAR_RS232_BASEADDR + 4,'Y');
for(h=0; h<delay_count/4000; h++);
XIo_Out32(XPAR_RS232_BASEADDR + 4,((av_Y>>8) & 0xF) + '0');
XIo_Out32(XPAR_RS232_BASEADDR + 4,((av_Y>>4) & 0xF) + '0');
XIo_Out32(XPAR_RS232_BASEADDR + 4,(av_Y & 0xF) + '0');
for(h=0; h<delay_count/4000; h++);
XIo_Out32(XPAR_RS232_BASEADDR + 4,'!');
for(h=0; h<delay_count/4000; h++);

//Servos
if (count > 30){                        //if object fills more that 160 pixels, else do nothing
        if (av_Y > CY){
                duty_count1 = duty_count1 + 500*(av_Y-CY);
        }
        else if (av_Y <= CY){
                duty_count1 = duty_count1 - 500*(CY-av_Y);
        }
        if (CX > av_X){
                duty_count2 = duty_count2 + 250*(CX-av_X);
        }
        else if (CX <= av_X){
                duty_count2 = duty_count2 - 250*(av_X-CX);
        }
        if (duty_count1 >= 115000){
                duty_count1 = 115000;
        }
        else if (duty_count1 <= 40000){
```

```
                duty_count1 = 40000;
            }
            if (duty_count2 >= 120000){
                duty_count2 = 120000;
            }
            else if (duty_count2 <= 30000){
                duty_count2 = 30000;
            }

            //Transmit PWM  to Servos
            XIo_Out32(XPAR_TILT_BASEADDR + 0x14, duty_count1);          //Send Value to
Tilt
            XIo_Out32(XPAR_ROTATE_BASEADDR + 0x14, duty_count2);      //Send Value to
Rotate
            }
        }
}
```