Student Name: Anne Harmeson
TA : Louis Brandy
William Dubel
Max Koessik
Instructor. A. A Arroyo

University of Florida
Department of Electrical and Computer Engineering
EEL 5666 - Spring 2004
Intelligent Machines Design Laboratory

Final Report

Cyclops

A red-loving robot

Table of Contents

Abstract

Cyclops the robot loves red and will look around a room for a red object, to follow and love forever. The purpose of this report is to give a detailed description of how Cyclops was successfully designed and implemented. This report will include a summary of the platform, actuation, sensors, behaviors, and electronics used to make this project a success. This report may be of interest to someone using a web cam with Java in a robot or interfacing a robot with a laptop wirelessly.

Executive summary

Cyclops is red-tracking robot. He is designed to wait for a red object to appear, and then track that object forever. Cyclops contains an ATmega128 micro controller, two infrared proximity sensors, three front bump switches, a radio frequency transmitter and receiver pair, a laptop, a wireless web camera, two servos, two front wheels, and a back caster. Cyclops is made from balsa wood and glue.

The special sensor Cyclops uses is the wireless web cam, the XCam2 from X10.com. With the camera mounted on top of the robot, the system allows the robot to capture video data and process the data externally on a laptop. My goal is to use the XCam2 system to allow my robot to sense a bright color and then track it, much like the CMUcam does. The XCam2 system is special because the Laptop is a more powerful processing tool than the robot's micro controller or the CMUcam, and I get to write my own image processing code.

Introduction

Haven't you ever wanted a robot that follows you when you wear your red pants? This is exactly what Cyclops will do!

Integrated System

Cyclops's integrated system consists of two parts: the on-board components controlled directly by the ATmega128, and the off-board components controlled by the PC. The ATmega128 micro controller controls the two servos attached to wheels for mobility, two infrared (IR) proximity sensors for obstacle avoidance, and a radio frequency (RF) receiver with decoder to receive commands (sent by the laptop). On the PC side, there is the small homemade board attached to the parallel port. Parallel data from the PC is transformed to serial data by the encoder. The RF transmitter takes the serial data and turns it into a RF signal. Software running on the PC will control and have access to this port, controlling the data sent in the RF signal. Placing data on the port sends a command over the RF signal to be received by the robot. The Robot's RF receiver gives serial data to the decoder, which turns the data back into a parallel signal. The XCam2 is attached to the robot, although the image data is actually controlled by the PC. A video receiver is attached to the PC through a USB port.

In the obstacle avoidance behavior, the micro controller will control the movement and behavior of Cyclops by constantly checking the state of the IR and bump sensors.

In the tracking behavior, obstacle avoidance is turned off, and movement is entirely controlled by commands received from the PC. The wireless web camera sends the visual data to the receiver connected to the laptop. The laptop processes this data, and determines how the

robot should respond. The laptop will then send a message back to Cyclops via the RF signal. Cyclops constantly checks the state of the RF receiver to update his behavior. By moving the image processing out of the micro controller and into the laptop, the data is processed much faster, and you have total control over the processing. By implementing the communication wirelessly, the robot does not have to handle the weight of an onboard laptop.

Mobile Platform

The objective of the platform is to allow the robot to look good and move around easily. I wanted the platform to be small and encase all the ugly components of the system. The only visible components I wanted were the XCam2, RF antenna, and the LCD (note: antennas are so cute, I strongly recommend using one in a robotics project).

The platform is made from the class-supplied balsa wood. The platform design is shown in Figure 1, with panels labeled and the key dimensions given. The width is 1/8'' thick. The IR sensors and LCD homes are specified as well. Inside the body lies a homemade board with the micro controller, an 8-pack of batters, and servos in the Lower Bottom component. The second battery pack for the camera is in the Bottom component. Figure 2 shows the assembled platform with all components mounted.

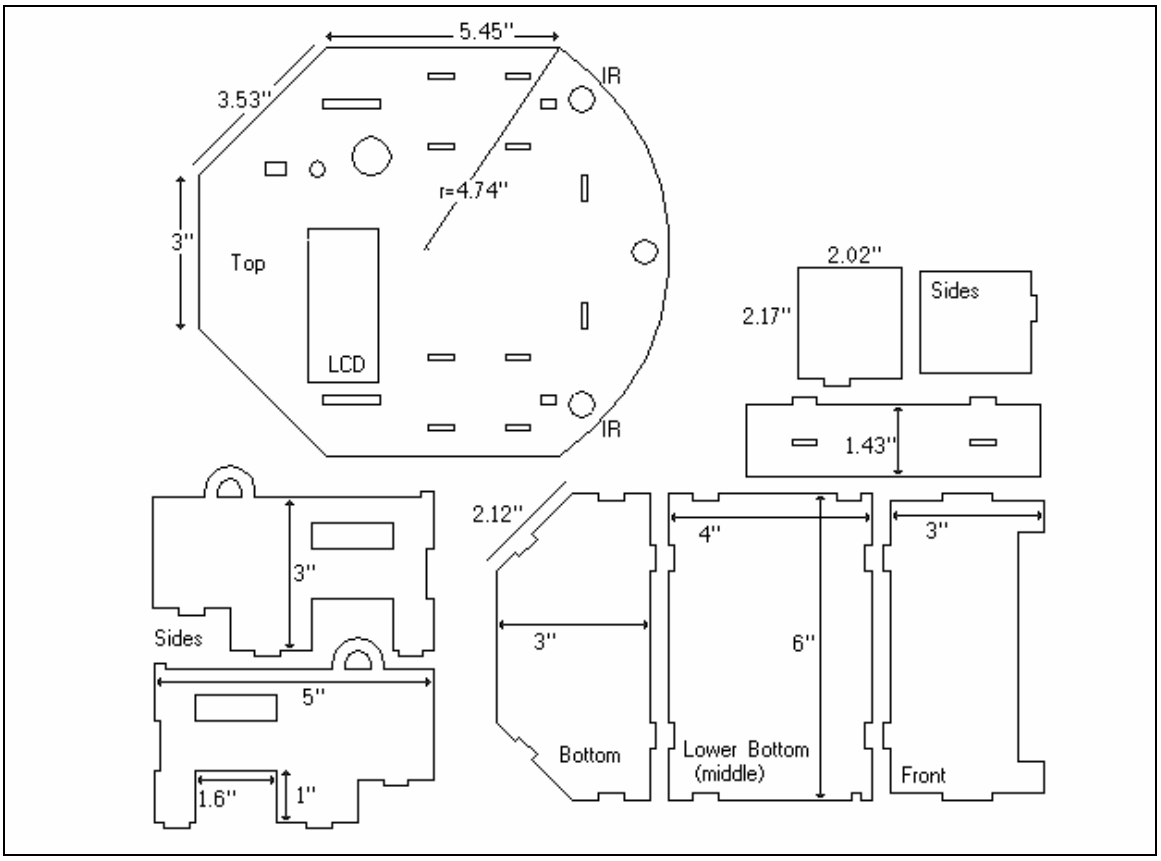


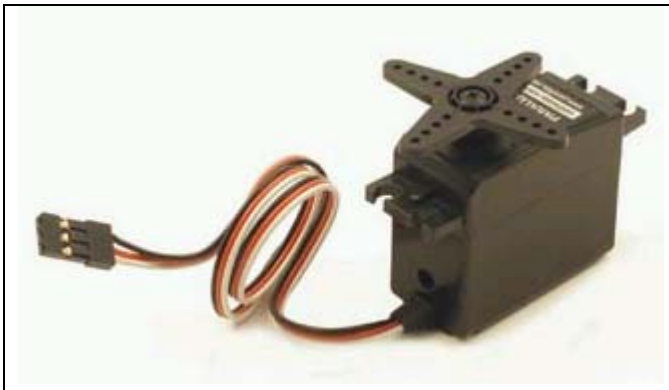
Figure 1: AutoCAD design of Cyclops platform.

Figure 2: Cyclops assembled platform.

Actuation

Cyclops is designed to move around the floor. He uses two continuous motion servos made by Parallax, and a back caster for stability. The servos and wheels were purchased from the Acroname website, the caster was purchased from Lowe's. Figure 3 is a picture of the servo

and some of its characteristics, both taken from the Acroname website. The servos are controlled with a PWM by the micro controller. The servos provide the ability to turn, move forward, and reverse. The PWM setup and servo control code is located in the AVR code section of the Appendixes.



Dimensions:	40.5 x 37.9 x 19.7 mm
Weight:	45.0 g
Output Torque:	3.4 kg-cm
Operating Speed at 4.8v	0.23 sec/60 degrees
Power Consumption:	6.0v/12mA at idle

Figure 3: Servo Information.

Sensors

The sensors on Cyclops are used for obstacle avoidance and to track a red object. Below is a summary of each sensor used and where it was purchased

IR

Two Sharp GP2D12 Detector Packages were used for as proximity sensors, mounted in the front with a crisscross view. The IR sensors were purchased from the Acroname website. I found that the cross sensor design could leave blind spots. This was overcome by writing a special IR calibration routine to store different threshold values for front and side obstacles. After using the calibration routine, the cross-view IR sensors are amazing and Cyclops hardly ever misses an obstacle. My IR setup and obstacle avoidance code is located in the AVR Code

section of the Appendixes. Figure 4 is a picture of the Detector Package taken from the Acroname website.

Figure 4: Sharp GP2D12 Detector Package.

Bump

Three bump switches are mounted in the front and front-sides with a bump skirt. These are designed to stop Cyclops if the IR sensors fail to detect an obstacle. Originally, I had hard-coded threshold values for the IR sensors in my obstacle avoidance code – creating a need for a backup system. This code did not always work because front obstacles and side obstacles needed different threshold values. I added bump switches to help out the IR, but then later I added a calibration routine to give the IR better values. After calibrating the robot, the IR always works, and the bump switches are not really necessary, but the skirt looks really good. My code for the bump switches is located in the ARV Code section of the Appendixes.

RF

The RF transmitter/receiver combination provides communication between my laptop and Cyclops. An RF signal is only sent in one direction: from the laptop to the robot. A two-way connection is not necessary since the laptop receives video data from the wireless web camera. The RF receiver is setup on the robot, and the RF transmitter is setup and on the laptop.

The following RF components were purchased from the Rentron website, the remote Control Store section:

- Transmitter (TXLC-434 434MHz High Performance RF TX Module)
- Receiver (RXLC-434 434MHz High Performance RF RX Module)
- Two antennas (50-OHM Whip Style Antenna)
- 8-bit encoder (Holtek HT-640 8-Bit Encoder IC)
- 8-bit decoder (Holtek HT-648L 8-Bit Decoder IC)

The RF works extremely well; I had no problems with distance. An encoder/decoder setup provides error protection and I had no problems with invalid transmissions. Figures 5 and 6 show the circuits used to implement the Transmitter (TX) and Receiver (RX). The 8-bit encoder/decoder provides 2^8 different commands. I probably used only 15 commands. The error correction value on bits A9-0 ensures that a transmission is correct. The error correction value can be any value, but it must be the same value on both the encoder and decoder. The SN74LS244 on the TX-side is a chip containing 8 tri-state buffers to protect you from damaging pins on your parallel port.

The RF was very easy to put together. The TX and RX come with schematics showing exactly how to set it up. The hardest part was figuring out a way to write data to the parallel port. In order to access my parallel port I had to install the DriverLINX Port I/O, a driver to give

port access under Windows XP, and download some .dll files. A link to this site is the Documentation section of this report. The code used to access the port is located in the Java Code section of the Appendixes.

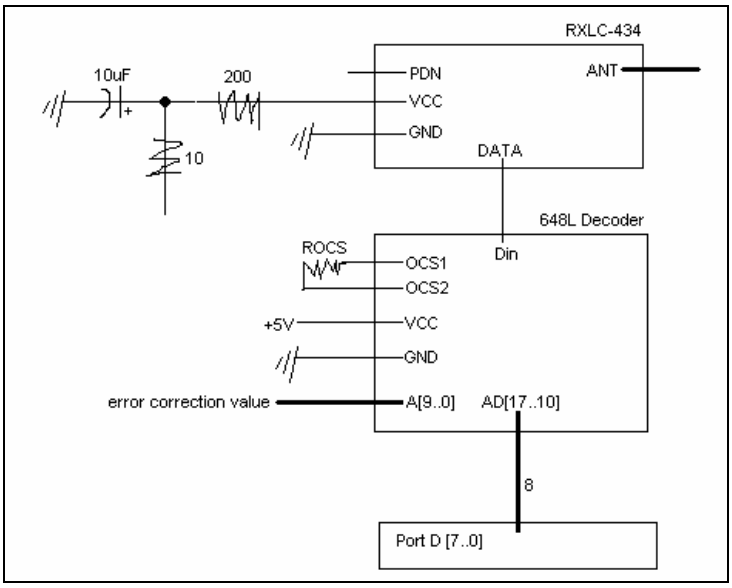


Figure 5: RX circuit.

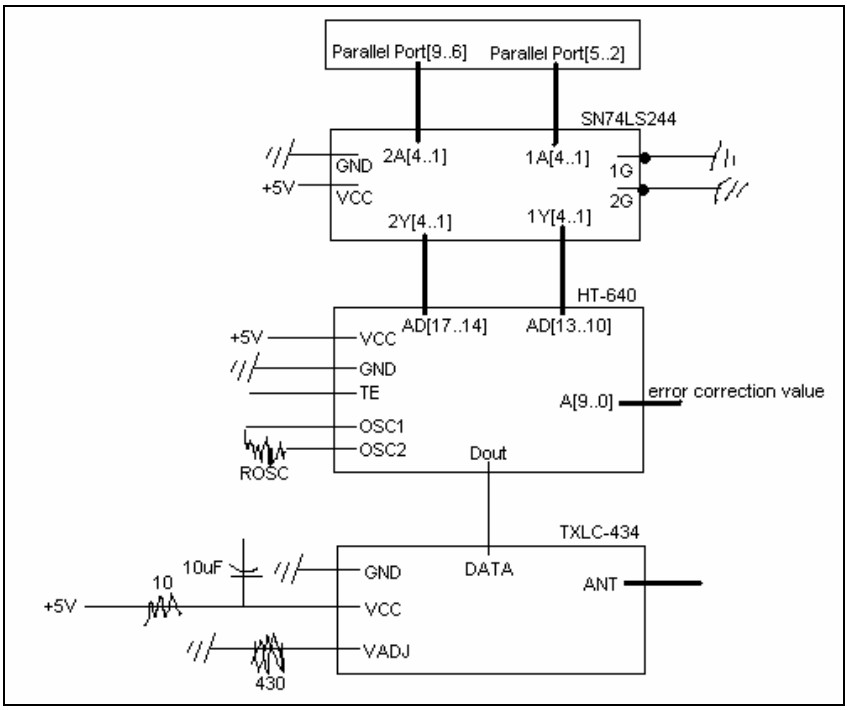


Figure 6: TX circuit

Xcam2 (Special Sensor)

The XCam2 system was purchased from X10.com. All the components are sold separately, but X10 offers a special wireless kit that includes a camera, receiver, and USB adapter for a special price of $130. In addition, X10 throws in a bunch of other wireless toys, because the kit is so special. I also received a Fire Cracker Computer Interface, Multi-Camera Remote, 2 Wireless Transceivers, and an XCam2 Battery Pack. Figure 7 is a picture of Camera, Receiver, and USB adapter, all taken from the X10 website.

Figure 7: XCam2, Video Receiver, USB Adapter

The kit comes with instructions on how to set up the camera on your computer and the USB driver information must be downloaded from the X10 website. The website also has a free program to view the captured video, XRay Vision, but it was of no use to my project. The wireless transmitter/receiver frequency of the camera system is 2.4 GHZ. The battery pack of the camera requires 4 AA batteries.

The XCam2 is mounted onto the robot, so that it can "see" facing forward. The video receiver with USB adapter plugs into my Laptop. My laptop is an IBM with the Windows XP

operating system. A Java application running on the Laptop captures the video data from the receiver. Another Java application analyzes the data and then produces intelligent commands to send back to the robot. Commands are sent back to the robot via RF sent through the laptop's parallel port.

I chose to develop the applications in Java and Windows because I am most familiar with the Java programming language and the Windows operating system. The Java 2 Platform Standard Edition, version 1.4, the Java Media Framework (JMF) version 2.1.1, and the Java Communications API are installed on my computer. The JMF API enables video-based media to be added to applications built on the Java technology. It can capture, playback, stream, and transcode multiple media formats. The Java Communications API contains support for IEEE 1284-parallel ports (not in Windows XP - see Conclusion section for details). Figure 8 shows a diagram of the communication structure of the XCam2 system.

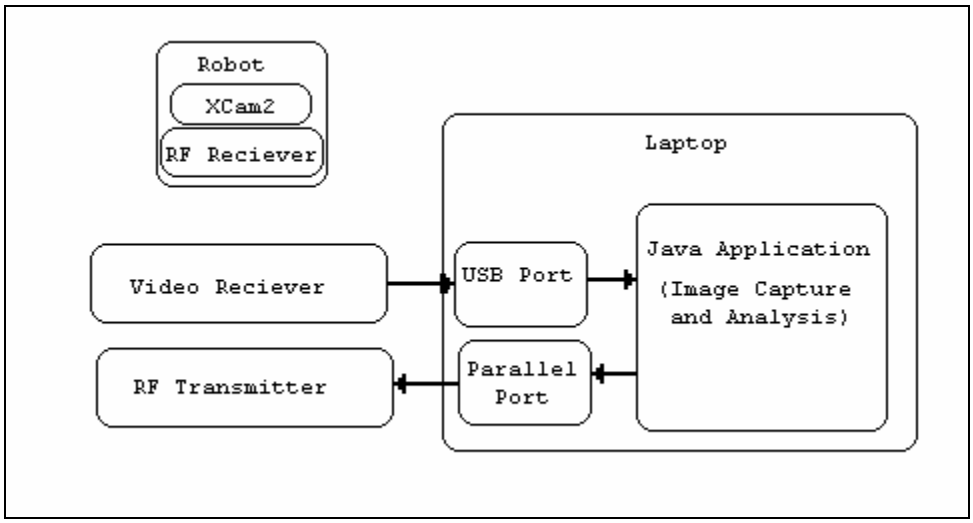


Figure 8: Sensor Integration Diagram

Since the wireless camera system uses a 2.4GHz frequency, the system can get interference from cordless phones and wireless Internet routers. The camera and receiver have channel select settings to allow you to pick a channel with the least amount of interference.

Inadequate lighting can also be a problem, causing the images to appear dark and very gray. I found that interference and lighting were a big problem at home, but never a problem in the IMDL lab.

Behaviors

The image analysis code supports tracking the largest, most red object in Cyclops's sight. Figure 9 shows a flow chart of the code. The actual code used may be found in the Java Code section of the Appendixes.

Each frame is divided into d columns; I chose a d-value of around 30. Each pixel of every frame is then traversed to find the column with the maximum red mean value (maxCol). This value is saved and used to determine a direction command for Cyclops. When there is a maxCol, a queue of previous maxCol's is created with always the last n recent maxCol's. This creates a memory of where the red was in case the object is lost in the next frame. It also helps to filter out noise by ignoring the current maxCol if its distance is too far away from the average of the queue. This system was not perfect, but I found it worked better than only checking the previous frame's maxCol. Determining the size of the queue depends on the frame rate. If the frame rate is low you will want a smaller queue or no queue at all because the averaging will not be as meaningful if the object can move across the whole frame in-between frames.

If there is no red in the current frame, then the queue is looked at next. If the queue size is greater than zero (there was red in a previous frame), then the code uses the average of the queue values to determine the side the red object was lost on. There is also a variable that keeps track of how many frames the red is lost - FramesNoRed. A command to circle is sent in attempt

to try to find the object on the side it was lost. The FramesNoRed variable determines how long this circle command will be sent before the robot determines it has lost the object completely, and stops. You could send this command out forever, knowing you lost the object on a particular side, but if the object was truly gone it might look dumb if the robot kept looking for too long.

If there is no red in a current frame, and the queue size is 0, or FramesNoRed has reached its limit, then the robot has lost the red object and is sent a command to stop and wait. The robot will resume the tracking behavior when a new red object is in sight.

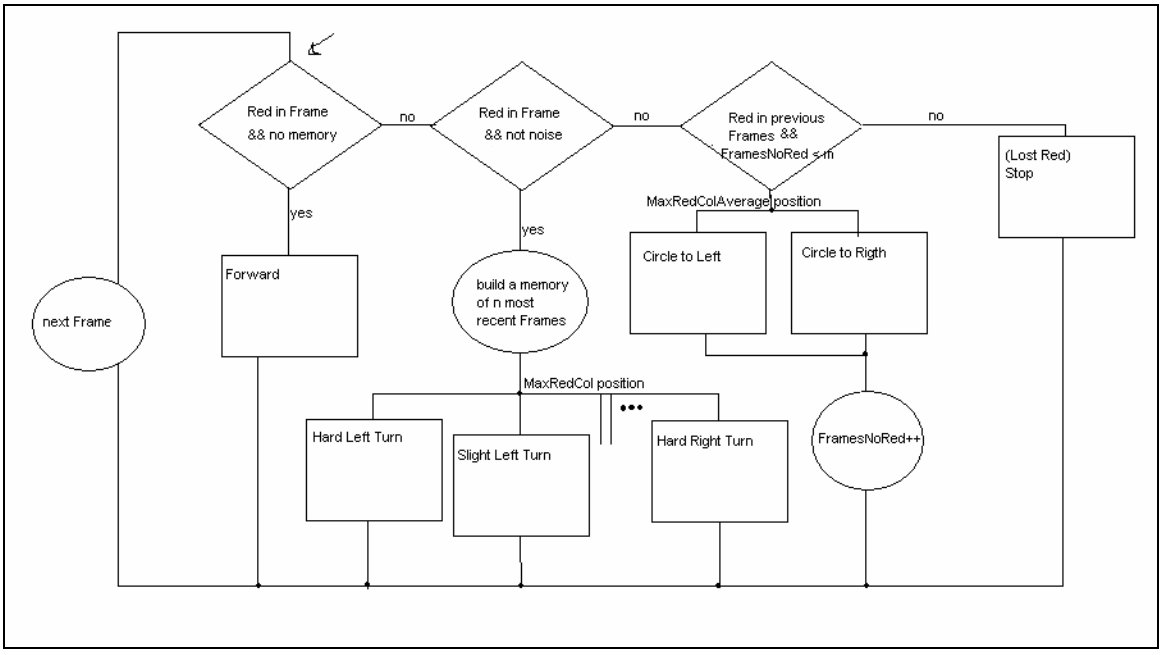


Figure 9: Flow chart of Cyclops Track Behavior

Experimental Layout and Results

In order to gain access to my parallel using Java and Windows, I had to install the DriverLINX Port I/O. I also had to wrap the DLL in another DLL using the Java Native

Interface. (This is a solution I found from Cyrus Harrison. If you want to learn more about the process there is a link to his project in the References section.) Somewhere in the installation of the DriverLINX, the changes caused my previous image analysis code to crash the Java Virtual Machine every time I tried to compile. This should never happen in Java, and it was very frustrating. Now I had two, wonderful, working pieces of code that were completely incompatible (and only days before demo-day).

Yet, I did have two versions of Java installed on my computer, and it was suggested by the TA's that maybe I could run both programs separately, in different Virtual Machines, and read and write a common file to pass information. This probably would have worked, but synchronizing the read/write methods seemed annoying. The solution I chose is random, weird, and I do not think it was the most eloquent. I had just finished a huge networking client-server project, and because the code and knowledge was available, I turned my parallel-port-command-sending application into a "Server", and my image-analysis-red-tracking application into a "Client". The server is even multi threaded to handle multiple clients and client disconnections without crashing – if that were ever necessary. The two applications communicate through a TCP connection.

The Server is the application that is responsible for sending commands to Cyclops by writing data to the parallel port, and constantly loops waiting for a message from the Client. The client is the application that analyzes the video data and creates intelligent commands. It is able to send these intelligent commands to the robot via the Server and the TCP connection. All was solved.

The RGB values for determining a red pixel and other variables in the red-tracking algorithm depend a lot on the robot's environment. The room lighting affects the RGB values,

and often I would have to re-determine the values in a new environment. I set thresholds in my Java application; a pixel was tested to see if it the red was greater than a minimum value, and if green and blue were less than a maximum-value. To improve the color definition, I would create a calibration method to make the application set appropriate thresholds, rather than performing guess and check myself.

Because my frame rate was fairly high, I was able to use a queue to generate an average of previous frames. This is important because the camera can pick up noise as red, or see multiple red objects in a room. I didn't want Cyclops twitching back and forth between red on opposite sides of the room. When I tried using only one frame for the memory, the tracking was not as good I found that only one previous frame did not provide enough information. I used a queue of size 10.

The variable FramesNoRed is determined by how fast the robot moves. If the robot turns very sharply, the sharp-turn command does not need to be sent as long because the robot will re-find the object faster. Although, if the robot circles too fast, it often misses the red object again because the object is not in sight for enough frames to accurately determine where the object is. This caused a lot of over correction, loosing the object again, and lots of circle turning. To avoid some of this behavior, I slowed the sharp-turn speed and added another variable. If I was sending the command to circle because the red object was lost, and in the next frame I re-found the object, then instead of determining a command based on the column position like usual, I would send a forward command instead to stop the turn and try to build a memory queue.

Finding the right code setup was interesting because modifications in the robot micro controller code affected what was necessary in the image analysis code, and visa versa. Finding

the right setup meant sometimes modifying the micro controller code, or the image analysis code, or both.

Conclusion

The most troublesome part of the project was using the Windows XP operating system. Although I had been advised that using Linux would be simple, better for accessing the camera data, and better for sending commands through the parallel port, I did not think it would be impossible in Windows. Windows XP is especially bad because it has built in security features to keep you from accessing your parallel port.

I originally thought I could use the Java COMM API to send data to my parallel port, but this did not work. Although Java advertises that the COMM API supports Parallel Port access, I did not ever witness this. Even the Java sample programs did not work correctly. Data was never actually put onto the parallel port; it only spooled in my printer queue. I found a lot of "solutions" on the Internet, but I did not have any success with them either. Thankfully, Cyrus Harrison had wanted to access his parallel port through Java in a Summer 2002 project (RANZOR) and I was able to use his code and install the driver that he found.

Overall, this was an amazing class. I have never put so much work into one project, but I had so much fun.

Documentation

Sources for Parts

Acroname: http://www.acroname.com/index.html
- Parallax Cont Rotation Servo
- Red Wheel with Black Band
- Sharp GP2D120 IR Sensor

E-Plus: Gainesville
- Various stuff

Lowe's: Gainesville
- Small Caster
- Various Stuff

Radio Shack: Gainesville
- Various stuff

Rentron: http://www.rentron.com/PicBasic/RemoteControl.htm
- Transmitter (TXLC-434 434MHz High Performance RF TX Module)
- Receiver (RXLC-434 434MHz High Performance RF RX Module)
- Two antennas (50-OHM Whip Style Antenna)
- 8-bit encoder (Holtek HT-640 8-Bit Encoder IC)
- 8-bit decoder (Holtek HT-648L 8-Bit Decoder IC)

Spark Fun Electronics: http://www.sparkfun.com/shop/index.php?shop=1&cart=56589&cat=1&
- ATMega128 Mini Header Board
- Parallel Port Dongle Programmer for STK Port

X10: www.X10.com
- XCam2 package: XCam2, Video Receiver, USB adapter, and batter pack

References:

TA's: Louis Brandy, William Dubel, Max Koessik
Instructor: A. A Arroyo

David Fischer's Java Programming Examples: Capture Video from Logitech QuickCam Pro 3000 Camera with Java JMF.
http://www.mutong.com/fischer/java/usbcam/

Interfacing the Standard Parallel Port.
http://www.beyondlogic.org/spp/parallel.htm#2

Java Communications API by Sun Microsystems, Inc.
http://java.sun.com/products/javacomm/index.jsp

Java Media Framework API Guide.
http://java.sun.com/products/java-media/jmf/2.1.1/guide/index.html

JMF API and Sample Code by Sun Microsystems, Inc.
http://java.sun.com/products/java-media/jmf/index.jsp

Parallel Printer Port Access through Java.
http://www.geocities.com/Juanga69/parport/

Port IO - Simple user mode DLL and NT kernel driver to allow direct hardware I/O access. Works under Windows 95, 98, Me, 2000, and XP. This software is unsupported.
http://www.driverlinx.com/
(Port IO Driver)
http://www.geocities.com/dinceraydin/python/indexeng.html
(winioport module for Python: winioport.zip)

Programming in Java Advanced Imaging.
http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/JAITOC.fm.html

Program Multimedia with JMF Tutorial by Budi Kurniawan.
http://www.javaworld.com/jw-04-2001/jw-0406-jmf1.html

RANZOR: IMDL robot by Cyrus Harrison.
http://www.mil.ufl.edu/imdl/papers/IMDL_Report_Summer_02/harrison_cyrus/ranzor.pdf

Rodriguez, Uriel: A Genius.

Appendix A: AVR Source Code

IR Code

```
/*********************************************
* Title:          ACD.h
* Programmer:     Anne Harmeson
* Date:           2/16/2004
* Version:        2
*
* Description:
*                 Code to use the Analogue to Digital Converter.

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

/*.................Includes.....................*/
#include <avr/io.h>
#include <stdio.h>

void ADC_init(void);
void ADC_selectChannel(uint8_t channel);
uint8_t ADC_read(void);
uint8_t ADC_sample(uint8_t channel);      //channel: 0-7
uint8_t ADC_sampleN(uint8_t channel, uint8_t n);
```

```
/*********************************************
* Title:          ACD.c
* Programmer:     Anne Harmeson
* Date:           2/16/2004
* Version:        2
*
* Description:
*                 Code to use the Analogue to Digital Converter.

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

#include "ADC.h"

/*************  Initialization:  ********************
*     set Port F (ADC PORT) as inputs
*     set ADCSR = '1000 0000' or '0x80'
*           (enable ADC, single conv. mode, no interrupt, no prescalar)
*********************************************/
void ADC_init(void)
{
      //configure ADC port (PORTF) as input
       DDRF  = 0x00;
```

```
      //outp(0x86, ADCSR);
      ADCSR = 0x80;
}
uint8_t ADC_sample(uint8_t channel)
{
      ADC_selectChannel(channel);
      return ADC_read();
}

uint8_t ADC_read(void)
{
      uint8_t value = 0;
      sbi(ADCSR,ADSC);        //start conversion
      loop_until_bit_is_set(ADCSR,ADIF);  //wait till conversion completes
      value = ADCH;           //read value
      sbi(ADCSR,ADIF);        //reset ADCI flag
  return value;
}

void ADC_selectChannel(uint8_t channel)
{
      /* select channel */
      ADMUX = 0x60; //AVCC with external capacitor at AREF pin, left adjusted
      ADMUX |= channel;
}
```

Bump Switch Code

```
/*********************************************
* Title:          Bump.h
* Programmer:     Anne Harmeson
* Date:           3/30/2004
* Version:        1
*
* Description:
*                 Code to initialize and use bump switches on Port C.
*
* Bump_PORT0 - callibration button  "0000 0001"
* Bump_PORT1 - front bump-skirt     "0000 0010"
* Bump_PORT2 - right bump-skirt     "0000 0100"
* Bump_PORT3 - left bump-skirt      "0000 1000"
* Bump_PORT4 -
* Bump_PORT5 -
* Bump_PORT6 -
* Bump_PORT7 -
*
*
*********************************************/

/*.................Includes.....................*/
#include <inttypes.h>
#include <avr/io.h>
/*.................end of Includes.................*/

/*.................Constants.....................*/
```

```
#define BUMP_PORT PORTC
#define BUMP_PIN PINC
#define BUMP_DDR DDRC
#define calibrateBump   0x01
#define frontBump 0x02
#define rightBump 0x04
#define letfBump  0x08
/*.................end of Constants.................*/

void bump_init(void);
void bump_setDDR(void);
uint8_t sampleBumpSwitches(void);
int getBump(uint8_t theBumpVal);
void bumpDB(void);
```

```
/*********************************************
* Title:          Bump.c
* Programmer:     Anne Harmeson
* Date:           3/30/2004
* Version:        1
*
* Description:
*                 Code to initialize and use bump switches on Port C.
*
* Bump_PORT0 - callibration button  "0000 0001"
* Bump_PORT1 - front bump-skirt     "0000 0010"
* Bump_PORT2 - right bump-skirt     "0000 0100"
* Bump_PORT3 - left bump-skirt      "0000 1000"
* Bump_PORT4 -
* Bump_PORT5 -
* Bump_PORT6 -
* Bump_PORT7 -
*
*********************************************/

#include "Bump.h"

void bump_init(void)
{
      bump_setDDR();
      BUMP_PORT = 0x00; //disable pullup resistor
}

void bump_setDDR(void)
{
      BUMP_DDR = 0x00; //set as inputs
}

uint8_t sampleBumpSwitches(void)
{
      //returns an 8-bit num. - ones corresponded to the button pushed
      //switches are active low!
      uint8_t bumpVals = BUMP_PIN;
      bumpVals = ~(bumpVals); //invert bits
      bumpVals &= 0x0F; //zero upper nibble
```

```
	bumpDB();   //switch debounce
	return bumpVals;
}

int getBump(uint8_t theBumpVal)
{
	//ie:  int i = getBump(frontBump)
	//1=true, 0=false
	uint8_t bumpVals = sampleBumpSwitches();
	bumpVals &= theBumpVal;
	if(bumpVals == theBumpVal)
	{
		return 1;
	}
	else
	{
		return 0;
	}
}

void bumpDB(void) //debounce?
{
	uint16_t time1;
		for(time1 = 0; time1 < 2000; time1++);
}
```

LCD Code

```
/***********************************************
* Title:          LCD.h
* Programmer:     Anne Harmeson
*                 Author: Max Billingsley
*                 Adapted by: David Wynacht
*                 Adapted again by me
* Date:           2/16/2004
* Version:        2
*
* Description:
*                 Code to initialize and use the LCD display.
*
* LCD_PORT0 - DB4
* LCD_PORT1 - DB5
* LCD_PORT2 - DB6
* LCD_PORT3 - DB7
* LCD_PORT4 - RS  = 0x10
* LCD_PORT5 - r/w = 0x20
* LCD_PORT6 - EN  = 0x40
*
* RS: Register Select
* 0 - Command Register
* 1 - Data Register
*
***********************************************/
```

```
/*.................Includes.....................*/
#include <inttypes.h>
#include <avr/io.h>
/*.................end of Includes.................*/

/*.................Constants.....................*/
#define LCD_PORT PORTA
#define     LCD_DDR     DDRA
#define     RS          0x10  // RS Signal "0001 0000"
#define     RW          0x20  // RW Signal
#define     ENABLE      0x40  // ENABLE Signal
/*.................end of Constants................*/

//Method Signatures
void LCD_setDDR(void);
void LCD_init(void);
void LCD_delay(void);
void LCD_delayLong(void);
void LCD_sendString(char *s);
void LCD_sendByte(uint8_t val);
void LCD_sendCommand(uint8_t val);
void LCD_home(void);
void LCD_clearScreen(void);
```

```
 /*********************************************
* Title:          LCD.c
* Programmer:     Anne Harmeson
*                 Author: Max Billingsley
*                 Adapted by: David Wynacht
*                 Adapted again by me
* Date:           2/16/2004
* Version:        3
*
* Description:
*                 Code to initialize and use the LCD display.
*
* LCD_PORT0 - DB4
* LCD_PORT1 - DB5
* LCD_PORT2 - DB6
* LCD_PORT3 - DB7
* LCD_PORT4 - RS  = 0x10
* LCD_PORT5 - r/w = 0x20
* LCD_PORT6 - EN  = 0x40
*
* RS: Register Select
* 0 - Command Register
* 1 - Data Register
*
*************************************************/

#include "lcd.h"

void LCD_setDDR(void)
{
      LCD_DDR = 0xff;
```

```
}

void LCD_init(void)
{
	//initialize the DDR
	LCD_setDDR();
	LCD_delay();

	LCD_sendCommand(0x33);		//enable 4-bit mode
	LCD_sendCommand(0x32);
	LCD_sendCommand(0x2c);		//enable 2-line mode
	LCD_sendCommand(0x0f);		//display, cursor, blink
	LCD_sendCommand(0x01);		//clear home
	//ready to write data to LCD
}

void LCD_delay(void)
{
	uint16_t time1;
	for(time1 = 0; time1 < 2000; time1++);
}

void LCD_delayLong(void)
{
	uint16_t i;
	uint16_t k;

	uint16_t var1 = 0;
	for (i = 0; i < 2000; i++)
	{
		for (k = 0; k < 2000; k++)
		{
			var1 = 0;
		}
	}
}

void LCD_sendCommand(uint8_t val)
{
	uint8_t lowerNibble = val;
	lowerNibble &= 0x0f;
	val >>= 4;

	LCD_PORT = val;
	LCD_delay();
	LCD_PORT |= ENABLE;
	LCD_PORT &= ~ENABLE;
	LCD_delay();

	LCD_PORT = lowerNibble;
	LCD_delay();
	LCD_PORT |= ENABLE;
	LCD_PORT &= ~ENABLE;
	LCD_delay();
}

void LCD_sendString(char *s)
```

```
{
      while (*s) LCD_sendByte(*s++);
}

void LCD_sendByte(uint8_t val)
{
      uint8_t lowerNibble = val;
      lowerNibble &= 0x0f;
      lowerNibble |= RS;      // set data mode

      val >>= 4;        //upper Nibble
      val |= RS;        //set data mode/


      LCD_PORT = val;
      LCD_delay();
      LCD_PORT |= ENABLE;
      LCD_PORT &= ~ENABLE;
      LCD_delay();

      LCD_PORT = lowerNibble;
      LCD_delay();
      LCD_PORT |= ENABLE;
      LCD_PORT &= ~ENABLE;
      LCD_delay();
}

void LCD_clearScreen (void)
{
      LCD_delay();
      LCD_sendCommand(0x01);
      LCD_delay();
}

void LCD_home(void)
{
      LCD_sendCommand(0x10);
}
```

Main Routine

```
/*********************************************
* Title:          Main.c
* Programmer:     Anne Harmeson
* Date:           3/31/2004
* Version:        1
*
* Description:    Main program tracking/obstacle avoidance
*                 listens to RF values for behavior
*                 Commented for only tracking behavior
*                 (obstacle avoidance and IR calibration commented out)
*********************************************/

#include <avr/pgmspace.h>
#include <stdio.h>
```

```
#include "LCD.h"
#include "ADC.h"
#include "Servos.h"
#include "Bump.h"
#include "RF.h"



#define CDS       ADC_sample(0);
#define IR_LEFT   ADC_sample(1);
#define IR_RIGHT  ADC_sample(2);
#define BUMP      sampleBumpSwitches();
#define RF_sample sampleRF();

#define calibrationBump 0x01
#define frontBump       0x02
#define rightBump       0x04
#define letfBump        0x08


/***************************** Globals *********************************/
      int rv, lv;

      //obstacle avoidance variables
      uint8_t distLeft;
      uint8_t distRight;
      uint8_t bumpVals;
      uint8_t tempBump;
      uint8_t RFval;

      //calibration variables
      uint8_t F_distLeft;
      uint8_t F_distRight;
      uint8_t R_distLeft;
      uint8_t R_distRight;
      uint8_t L_distLeft;
      uint8_t L_distRight;
/************************** End of Globals ******************************/

/************************** Method Declarations ************************/
void calibrateIR(void);
void bumpSkirtEval(void);
void decideBumps(uint8_t val);
void randomTurn(void);
void obstacleAvoid(void);
/********************** END of Method Declaration ***********************/


int main(void)
{
      //INITIALIZATIONS
      LCD_init();
      ADC_init();
      servos_init();
      bump_init();
      RF_init();
      fdevopen(LCD_sendByte,NULL,0);
```

```
	//TEST LCD DISPLAY
	printf("Cyclops!");
	LCD_delayLong();  //generic delay
	LCD_clearScreen();
	printf("Track RED!...");
	LCD_delayLong();  //generic delay

	//CALIBRATION
	//calibrateIR();

	RFval = RF_sample;
	LCD_clearScreen();
	printf("RF = %u",RFval);
	LCD_delayLong();  //generic delay

	LCD_clearScreen();


/************************************
RF Values

1=LEFT
2=RIGHT
4=FORWARD
8=REVERSE
16=HARD_RIGHT
32=HARD_LEFT
64=NO_RED

65=R3
66=R2
67=R1
68=L3
69=L2
70=L1

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


	//read RF forever -- track if red in sight, or obstacle avoid
	while (1)
	{
		servos_delay(10);

		RFval = RF_sample;
		//LCD_clearScreen();
		//printf("RF = %u",RFval);

		//distLeft = IR_LEFT;
		//distRight = IR_RIGHT;
		//bumpVals = BUMP;	//sample all bumpSwitches

		switch(RFval)
		{

			case(4): //forward
```

```
      RIGHT = RIGHT_FORWARD8;
      LEFT = LEFT_FORWARD8;
      //LCD_clearScreen();
      //printf("RF: FORWARD");
      //bumpSkirtEval();
break;

case(65): //R3
      RIGHT = RIGHT_FORWARD1;
      LEFT = LEFT_FORWARD8;

      //LCD_clearScreen();
      //printf("RF: Turn RIGHT (R3)");
      //bumpSkirtEval();
break;

case(66):  //R2
      RIGHT = RIGHT_FORWARD4;
      LEFT = LEFT_FORWARD8;

      //LCD_clearScreen();
      //printf("RF: Turn RIGHT (R2)");
      //bumpSkirtEval();

break;

case(67): // R1
      RIGHT = RIGHT_FORWARD8;
      LEFT = LEFT_FORWARD8;

      //LCD_clearScreen();
      //printf("RF: center (R1)");
      //bumpSkirtEval();
break;

case(68): // L1
      RIGHT = RIGHT_FORWARD8;
      LEFT = LEFT_FORWARD8;

      //LCD_clearScreen();
      //printf("RF: center (L1)");
      //bumpSkirtEval();
break;

case(69): // L2
      RIGHT = RIGHT_FORWARD8;
      LEFT = LEFT_FORWARD4;

      LCD_clearScreen();
      printf("RF: Turn LEFT (L2)");
      //bumpSkirtEval();
break;

case(70): // L3
      RIGHT = RIGHT_FORWARD8;
      LEFT = LEFT_FORWARD1;
```

```
                        //LCD_clearScreen();
                        //printf("RF: Turn LEFT (L3)");
                        //bumpSkirtEval();
                  break;

                  case(16): //HARD  RIGHT

                        RIGHT = RIGHT_REVERSE4;
                        LEFT = LEFT_FORWARD4;
                        //LCD_clearScreen();
                        //printf("RF: HARD right...");
                        //bumpSkirtEval();
                  break;

                  case(32):  //HARD_left
                        RIGHT = RIGHT_FORWARD4;
                        LEFT = LEFT_REVERSE4;
                        //LCD_clearScreen();
                        //printf("RF: HARD left...");
                        //bumpSkirtEval();
                  break;

                  case(64):  //NO red: obstacle avoid
                        LCD_clearScreen();
                        printf("No RED");
                        servos_stop();
                        //obstacleAvoid();
                  break;

                  default:  //error
                        LCD_clearScreen();
                        printf("Invalid RF Value: %u",RFval);
                        servos_stop();
                        //obstacleAvoid();
            }
  }   //end of while
}

void calibrateIR(void)
{
      LCD_clearScreen();
      printf("Calibrate Sensors...");
      LCD_delayLong();  //generic delay

      LCD_clearScreen();
      printf("FRONT obstacle...");
      LCD_delayLong();  //generic delay

      while(getBump(calibrationBump) == 0)
      {
            //wait for calibration button to be pushed
            //position robot with obstacle to RIGHT
      }

      F_distLeft = IR_LEFT;
      F_distRight = IR_RIGHT;
      LCD_sendCommand(0xC0);
```

```
	printf("R_IR= %u  L_IR= %u",F_distRight, F_distLeft);
	LCD_delayLong();  //generic delay

	LCD_clearScreen();
	printf("RIGHT obstacle...");
	LCD_delayLong();  //generic delay

	while(getBump(calibrationBump) == 0)
	{
		//wait for calibration button to be pushed
		//position robot with obstacle to RIGHT
	}

	R_distLeft = IR_LEFT;
	R_distRight = IR_RIGHT;
	LCD_sendCommand(0xC0);
	printf("R_IR= %u  L_IR= %u",R_distRight, R_distLeft);
	LCD_delayLong();  //generic delay

	LCD_clearScreen();
	printf("LEFT obstacle...");
	LCD_delayLong();  //generic delay

	while(getBump(calibrationBump) == 0)
	{
		//wait for calibration button to be pushed
		//position robot with obstacle to LEFT
	}

	L_distLeft = IR_LEFT;
	L_distRight = IR_RIGHT;
	LCD_sendCommand(0xC0);
	printf("R_IR= %u  L_IR= %u",L_distRight, L_distLeft);
	LCD_delayLong();  //generic delay

}//end of calibrateIR

void bumpSkirtEval(void)
{
		bumpVals = BUMP;  //sample all bumpSwitches
		tempBump = bumpVals & 0x0E; //evaluate only bump skirt

		//Ran into obstacle: Stop or Backup
		if(tempBump != 0x00 )
		{
			decideBumps(tempBump);
		}
}

void decideBumps(uint8_t val)
{
	//val =     "0000 LRF0"
	if(val == 0x0E)
	{
		drive_backward();
		LCD_clearScreen();
		printf("HIT: L, R, and F");
```

```
            LCD_sendCommand(0xC0);
            printf("Backup.");
            servos_delay(3000);
      }
      else if(val == 0x0C)
      {
            drive_backward();
            LCD_clearScreen();
            printf("HIT: L and R");
            LCD_sendCommand(0xC0);
            printf("Backup.");
            servos_delay(3000);
}
      else if(val == 0x0A)
      {
            drive_backward();
            LCD_clearScreen();
            printf("HIT: L and F");
            LCD_sendCommand(0xC0);
            printf("Backup ... Right.");
            servos_delay(3000);
            drive_right();
            servos_delay(3000);
      }
      else if(val == 0x08)
      {
            drive_backward();
            LCD_clearScreen();
            printf("HIT: L");
            LCD_sendCommand(0xC0);
            printf("Backup ... Right.");
            servos_delay(3000);
            drive_right();
            servos_delay(3000);
      }
      else if(val == 0x06)
      {
            drive_backward();
            LCD_clearScreen();
            printf("HIT: R and F");
            LCD_sendCommand(0xC0);
            printf("Backup ... Left.");
            servos_delay(3000);
            drive_left();
            servos_delay(3000);
      }
      else if(val == 0x04)
      {
            drive_backward();
            LCD_clearScreen();
            printf("HIT: R");
            LCD_sendCommand(0xC0);
            printf("Backup ... Left.");
            servos_delay(3000);
            drive_left();
            servos_delay(3000);
      }
```

```
      else if(val == 0x02)
      {
            drive_backward();
            LCD_clearScreen();
            printf("HIT: F");
            LCD_sendCommand(0xC0);
            printf("Backup ... Random-Turn.");
            servos_delay(3000);
            randomTurn();
      }
      else
      {
            servos_stop();
            LCD_clearScreen();
            printf("Anne is a LOSER");
            LCD_sendCommand(0xC0);
            printf("Stop!");
            servos_delay(3000);
            servos_delay(3000);

      }
}//end of bumpSkirtEval

void randomTurn(void)
{
      uint8_t num = distLeft | distRight;
      num = num & 0x01;

      if(num == 0x01) //turn right
      {
                  LCD_clearScreen();
                  printf("Random = RIGHT");
                  LCD_sendCommand(0xC0);
                  printf("R_IR= %u  L_IR= %u",distRight, distLeft);
                  drive_right();
                  servos_delay(3000);
      }
      else  //turn left
      {
                  LCD_clearScreen();
                  printf("Random = LEFT");
                  LCD_sendCommand(0xC0);
                  printf("R_IR= %u  L_IR= %u",distRight, distLeft);
                  drive_left();
                  servos_delay(3000);
      }
}


void obstacleAvoid(void)
{
            distLeft = IR_LEFT;
            distRight = IR_RIGHT;
            bumpVals = BUMP;  //sample all bumpSwitches

            //BumpSkirt Evaluation
            bumpSkirtEval();  //if bumped- driving L, R, or Backward
```

```
            if( (distLeft > F_distLeft) && (distRight > F_distRight)  )
            {
                  servos_stop();
                  LCD_clearScreen();
                  printf("Stop!");
                  LCD_sendCommand(0xC0);
                  printf("pick me up Anny");
                  servos_delay(3000);
                  servos_delay(3000);
            }
            //Front obstacle:  Backup, Random Turn
            else if ( (distLeft >= F_distLeft) || (distRight >= F_distRight)
)
            {
                  drive_backward();
                  LCD_clearScreen();
                  printf("Backward + randomTurn...");
                  LCD_sendCommand(0xC0);
                  printf("R_IR= %u  L_IR= %u",distRight, distLeft);

                  servos_delay(3000);
                  randomTurn();
            }
            //Tunnel condition: ?
            else if(0==1)
            {

            }
            //Left-side obstacle:  Turn Right
            else if( (distRight >= L_distRight) )
            {
                  drive_backward();
                  LCD_clearScreen();
                  printf("Backward + RIGHT...");
                  servos_delay(3000);
                  LCD_clearScreen();

                  drive_left();  //backwards code? right
                  LCD_clearScreen();
                  printf("Turn RIGHT");
                  LCD_sendCommand(0xC0);
                  printf("R_IR= %u  L_IR= %u",distRight, distLeft);
                  servos_delay(3000);
            }
            //Right-side obstacle:  Turn LEFT
            else if( distLeft >= R_distLeft)
            {
                  drive_backward();
                  LCD_clearScreen();
                  printf("Backward + LEFT...");
                  servos_delay(3000);
                  LCD_clearScreen();

                  drive_right();  ///backwards code //LEFT
                  LCD_clearScreen();
                  printf("Turn LEFT");
```

```
                LCD_sendCommand(0xC0);
                printf("R_IR= %u  L_IR= %u",distRight, distLeft);
                servos_delay(3000);
            }
            //No Obstacles: go FORWARD
            else
            {
                drive_forward();
                LCD_clearScreen();
                printf("Going FORWARD");
                LCD_sendCommand(0xC0);
                printf("R_IR= %u  L_IR= %u",distRight, distLeft);
                servos_delay(3000);

            }
}
```

RF Code

```
/*********************************************
* Title:          RF.h
* Programmer:     Anne Harmeson
* Date:           3/30/2004
* Version:        1
*
* Description:
*********************************************/

/*..................Includes.......................*/
#include <inttypes.h>
#include <avr/io.h>
/*..................end of Includes.................*/

/*..................Constants.......................*/
#define RF_PORT PORTD
#define RF_PIN PIND
#define RF_DDR DDRD
/*..................end of Constants................*/

void RF_init(void);
void RF_setDDR(void);
uint8_t sampleRF(void);
```

```
/*********************************************
* Title:          RF.c
* Programmer:     Anne Harmeson
* Date:           3/30/2004
* Version:        1
*
* Description:
*********************************************/

#include "RF.h"
```

```
void RF_init(void)
{
      RF_setDDR();
      RF_PORT = 0x00;   //disable pullup resisor
}

void RF_setDDR(void)
{
      RF_DDR = 0x00; //set as inputs
}

uint8_t sampleRF(void)
{
      return RF_PIN;
}
```

Servos Code

```
/*********************************************
* Title:          Servo.h
* Author:         Anne Harmeson
                  * Steven Pickles helped me a lot!
* Date:           2/21/2004
* Version:        1
*
* Description:
*                 Code to use Servos.

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

/*.................Includes.....................*/
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
/*.................end of Includes.................*/

/*
**    Motor Constants
*/
#define LEFT_FORWARD8 820
#define LEFT_FORWARD7 795
#define LEFT_FORWARD6 790
#define LEFT_FORWARD5 785
#define LEFT_FORWARD4 780
#define LEFT_FORWARD3 775
#define LEFT_FORWARD2 770
#define LEFT_FORWARD1 765
#define LEFT_STOP 760
#define LEFT_REVERSE1 750
#define LEFT_REVERSE2 745
#define LEFT_REVERSE3 740
#define LEFT_REVERSE4 735
#define LEFT_REVERSE5 730
#define LEFT_REVERSE6 725
```

```
#define LEFT_REVERSE7 720
#define LEFT_REVERSE8 700

#define RIGHT_FORWARD8 700
#define RIGHT_FORWARD7 720
#define RIGHT_FORWARD6 725
#define RIGHT_FORWARD5 730
#define RIGHT_FORWARD4 735
#define RIGHT_FORWARD3 740
#define RIGHT_FORWARD2 745
#define RIGHT_FORWARD1 750
#define RIGHT_STOP 757
#define RIGHT_REVERSE1 765
#define RIGHT_REVERSE2 770
#define RIGHT_REVERSE3 775
#define RIGHT_REVERSE4 780
#define RIGHT_REVERSE5 785
#define RIGHT_REVERSE6 790
#define RIGHT_REVERSE7 795
#define RIGHT_REVERSE8 820

#define LEFT OCR1A
#define RIGHT OCR1B

// Wheel Setup
void servos_init(void);

// High-Level Wheel Interface
void drive_forward(void);
void drive_backward(void);
void drive_right(void);
void drive_left(void);
void servos_stop(void);
void servos_delay(uint16_t ms);
```

```
/*************************************************
* Title:          Servo.c
* Author:         Anne Harmeson
* Date:           2/21/2004
* Version:        1
*
* Description:
*                 Code to use Servos.

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

#include "Servos.h"

void servos_init(void)
{

      outp(0xF0,DDRB);

      TCCR1A = 0xA0;
```

```
	//    Set Prescaler
	TCCR1B = 0x11;

	//set top value
	ICR1 = 10000;
	sei();
}


/*High Level Wheel Interface*/
void servos_stop(void)
{
	LEFT = LEFT_STOP;
	RIGHT = RIGHT_STOP;
}

void drive_forward(void)
{
	LEFT = LEFT_FORWARD8;
	RIGHT = RIGHT_FORWARD8;
}

void drive_backward(void)
{
	LEFT = LEFT_REVERSE8;
	RIGHT = RIGHT_REVERSE8;
}

void drive_left(void)
{
	LEFT = LEFT_REVERSE8;
	RIGHT = RIGHT_FORWARD8;
}

void drive_right(void)
{
	LEFT = LEFT_FORWARD8;
	RIGHT = RIGHT_FORWARD8;
}

void servos_delay(uint16_t ms)
{
	for(int x = 0; x < ms; x++)
	{
	  for(int i = 0; i<1000; i++); //1 ms
	}
}
```

Appendix B: Java Source Code

TrackRedCodec.java

```
package cyclops;

/**
 * <p> </p>
 * <p> </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p> </p>
 * @author Anne Harmeson, anne1@ufl.edu
 * @version 1.0
 */

import java.io.*;
import java.net.*;
import javax.media.*;
import javax.media.format.*;

import java.awt.*;
import java.awt.image.*;
import java.util.*;

public class TrackRedCodec implements Codec
{

  Format input, output = null;
  private Format supportedInFormat[] = new Format[]
                                      {new VideoFormat(null)};
  private Format supportedOutFormat[] = new Format[]
                                       {new VideoFormat(null)};

  private int width;
  private int height;

  private byte[] b; //array of RGB values from the video frame... wxhx3
  int frameNum = 0;
  PixelInterleavedSampleModel sm;
  DataBufferByte dataBuffer;
  DataBufferByte dataBufferNEW;
  Raster raster;

  //Client/Server variables
  private int portNumber;
  private String s;
  private Socket clientSocket;
  private DataOutputStream toServer;
  private BufferedReader inFromClient;
  private BufferedReader inFromServer;

  //Raster layer values
  private final int R = 0;
  private final int G = 1;
  private final int B = 2;

  //RF values
  private final byte RIGHT = (byte)1;
  private final byte LEFT = (byte)2;
  private final byte FORWARD = (byte)4;
  private final byte REVERSE = (byte)8;
  private final byte HARD_RIGHT = (byte)16;
  private final byte HARD_LEFT = (byte)32;
  private final byte NO_RED = (byte)64;
```

```
private final byte R3 = 65;
private final byte R2 = 66;
private final byte R1 = 67;

private final byte L3 = 68;
private final byte L2 = 69;
private final byte L1 = 70;

//vars to define red
private final int Rdef = 120;
private final int Gdef = 90;
private final int Bdef = 90;

int[] bandOffset =
    {2, 1, 0};

//colorColumn variables
int d = 30; //the number of columns in a frame
int maxCol; //col with max red mean value

//colorRow variables
int e = 30;
int maxRow;

//Column decision variables
double prevMaxRColMean = 0;
int prevMaxCol = -1;
LinkedList prevColQueue = new LinkedList(); //store the last x previous columns to average
int numFramesNoRedCol = 0;

//Row decision variables
double prevMaxRRowMean = 0;
int prevMaxRow = -1;
LinkedList prevRowQueue = new LinkedList(); //store the last x previous columns to average
int numFramesNoRedRow = 0;

//shared decision vars
int queueMaxSize = 10;
final int maxNumFramesNoRed = 100;
boolean ignoreQueue = false;

/**
 * Constructor
 * @param dim Dimension
 */
public TrackRedCodec(Dimension dim)
{
  width = dim.width;
  height = dim.height;
  sm = new PixelInterleavedSampleModel(0, width, height, 3, 3 * width, bandOffset);

  //client/server init
  portNumber = 7020;
  s = "anny";
  try
  {
    clientSocket = new Socket(s, portNumber);
    toServer = new DataOutputStream(clientSocket.getOutputStream());
    inFromClient = new BufferedReader(new InputStreamReader(System.in));
    inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
  }
  catch (IOException e)
  {
    System.out.println(e.getMessage());
    System.exit(0);
  }
}

public void sendCommand(String message)
{
  try
```

```
  {
    toServer.writeBytes(message + '\n');
  }
  catch (Exception ex)
  {
    System.out.println(ex.toString());
    System.exit(0);
  }
}

/**
 * accessFrame
 *
 * @param in Buffer
 */
public void accessFrame(Buffer inFrame)
{
  b = (byte[])inFrame.getData(); //length = w * h * 3
  frameNum++;

  //create Raster
  raster = Raster.createRaster(sm, (new DataBufferByte(b, b.length)), new Point());

  try
  {
    raster = redColumnRaster();
    raster = redRowRaster();
    updateQueue();
    //decide what to do
    redAnalysis2();
  }
  catch (Exception e)
  {
    System.out.println(e.toString());
  }

  //Covert raster back to an byte[]
  dataBufferNEW = (DataBufferByte)raster.getDataBuffer();
  b = dataBufferNEW.getData();
  inFrame.setData(b);
}

private int calcPrevMaxColAvg()
{
  int sum = 0;

  if (prevColQueue.size() == 0)
  {
    return -1;
  }
  for (int i = 0; i < prevColQueue.size(); i++)
  {
    sum += ( (Integer)prevColQueue.get(i)).intValue();
  }
  sum /= prevColQueue.size();
  return sum;
}

private int calcPrevMaxRowAvg()
{
  int sum = 0;

  if (prevRowQueue.size() == 0)
  {
    return -1;
  }
  for (int i = 0; i < prevRowQueue.size(); i++)
  {
    sum += ( (Integer)prevRowQueue.get(i)).intValue();
  }
  sum /= prevRowQueue.size();
```

```
    return sum;
  }

  /**
   * If red - locate and tell Cyclops where to go
   * try differentiate between bouncing red bands (2 competing red objects in frame)
   *
   */
  private void redAnalysis2()
  {
    int prevColAvg = calcPrevMaxColAvg();
    int diffCol = Math.abs(maxCol - prevColAvg);

    int prevRowAvg = calcPrevMaxRowAvg();
    int diffRow = Math.abs(maxRow - prevRowAvg);

    if (maxCol != -1 && ignoreQueue == true)
    {
      sendCommand(FORWARD + ""); //
    }
    else if (maxCol != -1 && (diffCol < d / 2))
    {
      //right (3)
      if ( (maxCol > 5 * d / 6))
      {
        System.out.println("RED screen-right (R3)");
        sendCommand(R3 + "");
      }
      else if ( (maxCol > 4 * d / 6))
      {
        System.out.println("RED screen-right (R2)");
        sendCommand(R2 + "");
      }
      else if ( (maxCol > 3 * d / 6))
      {
        System.out.println("RED screen-right (R1)");
        sendCommand(R1 + "");
      }
      else if ( (maxCol == d / 2))
      {
        System.out.println("RED screen-center");
        sendCommand(FORWARD + "");
      }
      else if ( (maxCol > 2 * d / 6))
      {
        System.out.println("RED screen-left (L1)");
        sendCommand(L1 + "");
      }
      else if ( (maxCol > 1 * d / 6))
      {
        System.out.println("RED screen-left (L2)");
        sendCommand(L2 + "");
      }
      else if ( (maxCol >= 0))
      {
        System.out.println("RED screen-left (L3)");
        sendCommand(L3 + "");
      }
    }
    /*
     else if ((prevRowQueue.size() > 0) && prevColQueue.size() > 0 && ! ( (prevColAvg > (3 * d /
4))) &&
      ! ( (prevColAvg < (d / 4)))) //lost on top?
       {
      if ( (prevRowAvg > (3 * d / 4)) && ! ( (prevColAvg > (3 * d / 4))) &&
      ! ( (prevColAvg < (d / 4))))
      {
      System.out.println("PREVIOUS top");
      System.out.println("Num Frames No Red = " + numFramesNoRedRow);
      sendCommand(FORWARD + "");
      }
```

```
    }
   */
  else if (prevColQueue.size() > 0) //no red,check previous frame
  {
    //System.out.println("*******************************");
    if ( (prevColAvg > (d / 2))) //lost on the right- hard right turn
    {
      System.out.println("PREVIOUS right");
      System.out.println("Num Frames No Red = " + numFramesNoRedCol);
      sendCommand(HARD_RIGHT + "");
    }
    else if ( (prevColAvg < d / 2)) //lost on left -- hard left turn
    {
      System.out.println("PREVIOUS left");
      System.out.println("Num Frames No Red = " + numFramesNoRedCol);
      sendCommand(HARD_LEFT + "");
    }
  }
  else // no red
  {

    System.out.println("PrevColQ length:  " + prevColQueue.size());
    System.out.println("no red");
    sendCommand(NO_RED + "");
  }
}


/**
 * Finds the column with the largest RED mean-value, creates a vertical
 * red band in that column to display to the screen, rest of the image is
 * unaltered.
 * @param d int
 * @return Raster
 */
private Raster redColumnRaster()
{
  double[] RColSum = new double[d];
  double[] RColMean = new double[d];
  int subWidth = width / d;
  WritableRaster rasterNEW = raster.createWritableRaster(raster.getSampleModel(),
                                                         raster.getDataBuffer(), new Point());

  //traverse raster for each pixil
  for (int multX = 0; multX < d; multX++)
  {
    for (int x = multX * subWidth; x < ( (multX * subWidth) + subWidth); x++)
    {
      for (int y = 0; y < height; y++)
      {
        //Definition of a red pixil
        if ( (rasterNEW.getSample(x, y, R) >= Rdef) && (rasterNEW.getSample(x, y, G) < Gdef) &&
             (rasterNEW.getSample(x, y, B) < Bdef))
        {
          RColSum[multX] += rasterNEW.getSample(x, y, R);
        }
      }
    } //end of x
  } //end of X multiplier

  //calc column means
  double numPixils = (subWidth * height);
  for (int x = 0; x < d; x++)
  {
    RColMean[x] = RColSum[x] / numPixils;
  }

  //find col# and value of the max RED-Column-Mean
  //prevMaxCol = calcPrevMaxColAvg();
  double maxRColMean = 0;
  maxCol = -1;
```

```
  for (int i = 0; i < d; i++)
  {
    maxRColMean = Math.max(maxRColMean, RColMean[i]);
    if (maxRColMean == RColMean[i])
    {
     maxCol = i;
    }
  }

  //no red in frame
  //if ( (maxRColMean < 1) || (maxCol == -1))
  //{
  //  maxCol = -1;
  //}
  //call in "access frame after the redRowRaster so that both can queues can update
  //updateColQueue(); //handles info about previous frames

  //no red in frame
  if ( (maxRColMean < 1) || (maxCol == -1))
  {
    maxCol = -1;
    return rasterNEW;
  }

  //otherwise, display the max colum to be all red
  //System.out.println("Max Red Col Mean = " + maxRColMean);
  for (int x = maxCol * subWidth; x < ( (maxCol * subWidth) + subWidth); x++)
  {
    for (int y = 0; y < height; y++)
    {
     rasterNEW.setSample(x, y, R, 255);
     rasterNEW.setSample(x, y, G, 0);
     rasterNEW.setSample(x, y, B, 0);
    }
  }

  //display previous max column
  if (prevMaxCol != -1 && prevMaxCol != maxCol)
  {
    for (int x = prevMaxCol * subWidth; x < ( (prevMaxCol * subWidth) + subWidth); x++)
    {
     for (int y = 0; y < height; y++)
     {
       rasterNEW.setSample(x, y, R, 255);
       rasterNEW.setSample(x, y, G, 255);
       rasterNEW.setSample(x, y, B, 0);
     }
    }
  }

  return rasterNEW;
}

private Raster redRowRaster()
{

  double[] RRowSum = new double[d];
  double[] RRowMean = new double[d];
  int subHeight = height / e;

  WritableRaster rasterNEW = raster.createWritableRaster(raster.getSampleModel(),
                                                         raster.getDataBuffer(), new Point());
  //do not display if column function did not detect red
  if (maxCol == -1)
  {
    maxRow = -1;
    return rasterNEW;
  }

  //traverse raster for each pixil
  for (int multY = 0; multY < e; multY++)
```

```
  {
    for (int y = multY * subHeight; y < ( (multY * subHeight) + subHeight); y++)
    {
     for (int x = 0; x < width; x++)
     {
       //Definition of a red pixil
       if ( (rasterNEW.getSample(x, y, R) >= Rdef) && (rasterNEW.getSample(x, y, G) < Gdef) &&
           (rasterNEW.getSample(x, y, B) < Bdef))
       {
         RRowSum[multY] += rasterNEW.getSample(x, y, R);
       }
     }
    } //end of y
  } //end of Y multiplier

  //calc row means
  double numPixils = (subHeight * width);
  for (int y = 0; y < e; y++)
  {
    RRowMean[y] = RRowSum[y] / numPixils;
  }

  //find row# and value of the max RED-Row-Mean
  double maxRRowMean = 0;
  maxRow = -1;
  for (int i = 0; i < e; i++)
  {
    maxRRowMean = Math.max(maxRRowMean, RRowMean[i]);
    if (maxRRowMean == RRowMean[i])
    {
     maxRow = i;
    }
  }

  //no red in frame
  /*
     if ( (maxRRowMean < 1) || (maxRow == -1))
     {
    maxRow = -1;
     }
   */
  //updateRowQueue(); //handles info about previous frames

  //no red in frame
  if ( (maxRRowMean < 1) || (maxRow == -1))
  {
    maxRow = -1;
    return rasterNEW;
  }

  //display the max Row to be all red
  //System.out.println("Max Red Row Mean = " + maxRRowMean);
  System.out.println("Red Row = " + maxRow);
  for (int y = maxRow * subHeight; y < ( (maxRow * subHeight) + subHeight); y++)
  {
    for (int x = 0; x < width; x++)
    {
     rasterNEW.setSample(x, y, R, 255);
     rasterNEW.setSample(x, y, G, 0);
     rasterNEW.setSample(x, y, B, 0);
    }
  }
  return rasterNEW;
}

/**
 * call after col and row raster
 */
private void updateQueue()
{
  //update column queue
```

```
  if (maxCol != -1)
  {
    if (numFramesNoRedCol > 5)
    {
     ignoreQueue = true;
    }
    else
    {
     ignoreQueue = false;
    }
    numFramesNoRedCol = 0;
    prevColQueue.addFirst(new Integer(maxCol));
    if (prevColQueue.size() > queueMaxSize)
    {
     prevColQueue.removeLast();
    }
  }
  else
  {
    numFramesNoRedCol++;
    if (numFramesNoRedCol > maxNumFramesNoRed)
    {
     prevColQueue = new LinkedList();
    }
  }

  //updata row queue
  if (maxRow != -1)
  {
    numFramesNoRedRow = 0;
    prevRowQueue.addFirst(new Integer(maxRow));
    if (prevRowQueue.size() > queueMaxSize)
    {
     prevRowQueue.removeLast();
    }
  }
  else
  {
    numFramesNoRedRow++;
    if (numFramesNoRedRow > maxNumFramesNoRed)
    {
     prevRowQueue = new LinkedList();
    }
  }
}

public static void main(String[] args)
{
  //ImageAnaysisCodec imageAnaysis1 = new ImageAnaysisCodec();
}

public Format[] getSupportedInputFormats()
{
  return supportedInFormat;
}

public Format[] getSupportedOutputFormats(Format parm1)
{
  if (parm1 == null)
  {
    return supportedOutFormat;
  }
  else
  {
    Format outs[] = new Format[1];
    outs[0] = parm1;
    return outs;
  }
}

public Format setInputFormat(Format parm1)
```

```
  {
    input = parm1;
    return input;
  }

  public Format setOutputFormat(Format parm1)
  {
    output = parm1;
    return output;
  }

  public int process(Buffer in, Buffer out)
  {
    accessFrame(in);

    // Swap the data between the input & output.
    Object data = in.getData();
    in.setData(out.getData());
    out.setData(data);

    // Copy the input attributes to the output
    out.setFormat(in.getFormat());
    out.setLength(in.getLength());
    out.setOffset(in.getOffset());

    return BUFFER_PROCESSED_OK;
  }

  public String getName()
  {
    return ("Red-tracking codec - by Anny");
  }

  public void open() throws javax.media.ResourceUnavailableException
  {}

  public void close()
  {
  }

  public void reset()
  {
  }

  public Object[] getControls()
  {
    return new Object[0];
  }

  public Object getControl(String parm1)
  {
    return null;
  }

}
```

CyclopsApp.java

```
package cyclops;

/**
 * <p> </p>
 * <p> </p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p> </p>
 * @author Anne Harmeson
 * copied ALOT from Cycrus Harrison's "Ranzor.java"
 * @version 1.0
 */
```

```
import java.io.*;
import javax.media.*;
import javax.media.control.*;
import javax.media.format.*;
import java.awt.*;
import java.awt.event.*;

public class CyclopsApp extends Frame implements ControllerListener, ActionListener,
    ItemListener
{
  private MediaLocator ml;
  private Processor videoProc;
  private TrackControl tc[] = null;
  private Object waitSync = new Object();
  private boolean stateTransitionOK = true;
  int height, width = 300;
  Dimension videoDim = new Dimension();

  /* capture url = vfw://0 */
  public CyclopsApp(String title, String url)
  {
    super(title);

    addWindowListener(new WindowAdapter()
    {
      public void windowClosing(WindowEvent we)
      {
       System.out.println("Closing Application");
       System.exit(0);
      }
    });

    //set up GUI
    makeGUI();
    //set up the Processor
    initProcessor(url);
  }

  public static void main(String[] args)
  {
    CyclopsApp processTest1 = new CyclopsApp("Cyclops!", "vfw://0");
  }

  /**
   * init
   */
  private void initProcessor(String capURL)
  {
    try
    {
      ml = new MediaLocator(capURL);
      videoProc = Manager.createProcessor(ml);
      videoProc.addControllerListener(this);
      videoProc.configure();
      if (!waitForState(videoProc.Configured))
      {
       System.err.println("Failed to configure the processor.");
       return;
      }
      videoProc.setContentDescriptor(null);

      tc = videoProc.getTrackControls();
      if (tc == null)
      {
       System.err.println("Failed to obtain track controls from the processor.");
       return;
      }

      TrackControl videoTrack = null;
      System.err.println(tc.length);
```

```
    for (int i = 0; i < tc.length; i++)
    {
      if (tc[i].getFormat() instanceof VideoFormat)
      {
        videoTrack = tc[i];
        System.err.println("VideoFormat (from tc[" + i + "]) is: " +
                           videoTrack.getFormat().toString());
        VideoFormat v = (VideoFormat)videoTrack.getFormat();
        videoDim = v.getSize();
        System.err.println("V_DIM: " + videoDim.toString());

        //break;
      }
    }
    if (videoTrack == null)
    {
      System.err.println("The input media does not contain a video track.");
      return;
    }

    // Instantiate and set the frame access codec to the data flow path.
    try
    {
      TrackRedCodec myCodec = new TrackRedCodec(videoDim);
      Codec codec[] = {myCodec};
      videoTrack.setCodecChain(codec);
    }
    catch (UnsupportedPlugInException e)
    {
      System.err.println("The process does not support effects.");
    }

    videoProc.realize();

    if (!waitForState(videoProc.Realized))
    {
      System.err.println("Failed to realize the processor.");
      return;
    }
  }

  catch (NoProcessorException ex)
  {
    System.out.println(ex.getMessage());
  }
  catch (IOException ex)
  {
    System.out.println(ex.getMessage());
  }
  // Start the processor.
  videoProc.start();
  setVisible(true);
}

private void makeGUI()
{
  //set up GUI
  show();
  setSize(width, height);
  System.out.println("set the size..");
  setLayout(new BorderLayout());
}

/**
 * Block until the processor has transitioned to the given state.
 * Return false if the transition failed.
 */
private boolean waitForState(int state)
{
  synchronized (waitSync)
  {
```

```
    try
    {
      while (videoProc.getState() != state && stateTransitionOK)
      {
        waitSync.wait();
      }
    }
    catch (Exception e)
    {
      System.err.println(e.getMessage());
    }
  }
  return stateTransitionOK;
}

/**
 * controllerUpdate
 *
 * @param controllerEvent ControllerEvent
 * @todo Implement this javax.media.ControllerListener method
 */
public void controllerUpdate(ControllerEvent controllerEvent)
{
  System.out.println(controllerEvent.toString());
  if (controllerEvent instanceof RealizeCompleteEvent)
  {
    synchronized (waitSync)
    {
      stateTransitionOK = true;
      waitSync.notifyAll();
    }
    Component comp;
    Dimension d = new Dimension();
    System.out.println("Adding visual component...");
    if ( (comp = videoProc.getVisualComponent()) != null)
    {
      add("Center", comp);
    }
    System.out.println("Adding control panel...");
    if ( (comp = videoProc.getControlPanelComponent()) != null)
    {
      d = videoProc.getControlPanelComponent().getSize();
      System.err.println("DimControl: " + d.toString());
      add("South", comp);
    }

    //calculate the frame dimensions to show video in correct size
    width = videoDim.width + getInsets().left + getInsets().right;
    height = videoDim.height + d.height + getInsets().top + getInsets().bottom;
    setSize(width, height);
    validate();
  }
  else if (controllerEvent instanceof ConfigureCompleteEvent ||
           controllerEvent instanceof PrefetchCompleteEvent)
  {
    synchronized (waitSync)
    {
      stateTransitionOK = true;
      waitSync.notifyAll();
    }
  }
  else if (controllerEvent instanceof ResourceUnavailableEvent)
  {
    synchronized (waitSync)
    {
      stateTransitionOK = false;
      waitSync.notifyAll();
    }
  }
  else if (controllerEvent instanceof EndOfMediaEvent)
  {
```

```
      videoProc.close();
      System.exit(0);
    }
  } //end of Controller Updata method

  /**
   * Invoked when an action occurs.
   *
   * @param e ActionEvent
   * @todo Implement this java.awt.event.ActionListener method
   */
  public void actionPerformed(ActionEvent e)
  {

  }

  /**
   * Invoked when an item has been selected or deselected by the user.
   *
   * @param e ItemEvent
   * @todo Implement this java.awt.event.ItemListener method
   */
  public void itemStateChanged(ItemEvent e)
  {
  }

}
```

Server.java

```
        import java.io.*;
        import java.net.*;
        import java.util.*;


//This is the main server class when we run server
//we run this classes main method
   public class Server
   {
      final int BUFSIZE = 1024;
      byte[] buffer = new byte[BUFSIZE]; // area for recv data
      final int timeout = 10000;
      SocketListener inThread; // this Server's listening thread
      ServerSocket socket = null;  //the main server socket

    //sets up the main operations of the server, starts the threads
      private void init()
      {
         try
         {
            socket = new ServerSocket(7020);
         }
            catch (SocketException e)
            {
               //problem createing socket
               System.err.println(e.getMessage());
               System.exit(0);
            }
            catch (IOException ex)
            {
               System.out.println(ex.getMessage());
            }

      // create listening thread
         while (true)
         {
            try
            {
            //call socket listener when we get a new connection
```

```
            inThread = new SocketListener(socket.accept(), this);
         //create a new thread with our socketListener object
            Thread t = new Thread(inThread);
         //start the thread
            t.start();
         }
            catch (IOException ex1)
            {
               System.out.println(ex1.getMessage());
            }
      }
   }//end init

 //returns the main server socket
   public ServerSocket getSocket()
   {
      return socket;
   }


 //create a new server and initalize it
   public static void main(String args[])
   {
      Server ourServer = new Server();
      ourServer.init();
   }

}
```

SocketListener.java
--contains PP info

```
import java.io.*;
import java.net.*;
import java.util.*;

class SocketListener implements Runnable
{
   Server parent;
   Socket client;
   boolean stop = false;
   private BufferedReader inFromClient;
   DataOutputStream outToClient;
   byte b = 0;
   PPort rf;

//constructor
   public SocketListener(Socket socket, Server server)
   {
      client = socket;
      parent = server;

      try
      {
         inFromClient = new BufferedReader(new
                              InputStreamReader(client.getInputStream()));
         outToClient = new DataOutputStream(client.getOutputStream());
         rf = new PPort();
         stop = false;
      }
         catch (IOException ex)
         {
            System.err.println(ex.getMessage());
            stop = true;
         }
   }
```

```
   /** run
   * continuously wait for commands form client(CyclopsAPP)
   */
      public void run()
      {
         while (!stop)
         {
            String command = getMessage();
            if (command != null)
            {
               try
               {
                  b = (byte)Integer.parseInt(command);
                  sendCommand(b);
               }
                  catch (NumberFormatException ex)
                  {
                     System.out.println(ex.toString());
                  }
            }
         } //end while
         System.out.println("Thread done.");

      } //end of run

      private synchronized String getMessage()
      {
         String message = null;
         try
         {
            message = inFromClient.readLine();
         }
            catch (IOException ex)
            {
               System.err.println(ex.getMessage());
               stop = true;
            }
         return message;
      }

      private synchronized void sendCommand(byte b)
      {
         //System.out.println("Command to send: " + b);
         rf.out(b);
                     System.out.println("Sent: " + b);
      }
   }
```

PPort.java

```
/*Simple Class to Wrap Parallel Port Access*/
// Cyrus Harrison (cdh@cise.ufl.edu)
   public class PPort
   {
      private dMem io;

      public PPort()
      {
         io = new dMem();
      }

      public void out(byte data)
      {
         io.write(0x3BC, (byte)data);
      }
   };
```

dMem.java

```
// JNI Wrapper Class Access Native Direct Memory IO
// Cyrus Harrison (cdh@cise.ufl.edu)
   class dMem
   {
   // Read from an address
      native public byte read(int Addr);

   // Write to an address
      native public void write(int Addr, byte Data);

   // Load the direct memory io driver wrapper class
      static
      {
         System.loadLibrary("dmemio");
      }
   }
```