Nigel

Brian Ruck

EEL 5666: Intelligent Machines Design Lab

Final Report

Instructor: Dr. Arroyo

TAs: Uriel Rodriguez

Jason Plew

Table of Contents

Abstract

Nigel is an autonomous candy man. This Final report is intended to give detailed information about Nigel's behaviors, functions, and construction. It was my goal to construct a fun and interesting robot that would perform certain behaviors that were both original and entertaining. This report will include a summary of Nigel's platform, sensors, actuation, and electronics used to make this project a success.

Executive Summary

Nigel is an Autonomous candy distributor. Nigel displays fun and interactive behaviors that prove to make the robot unique. Nigel has three modes of behavior; Track mode, Dispense Mode and Hide Mode. Each of these modes has a distinct function that characterized Nigel's personality. An Atmel AVR-Mega323 chip is used to control Nigel and all his characteristics.

In track mode, Nigel uses a CMUcam that detects the color blue. Nigel will first spin around looking for the color. If the color is not found he will roam around the room for a few seconds finding a new spot to track again. Once the object is found, Nigel will approach the object and go into dispense mode.

Dispense mode utilizes a servo that controls the candy being dispensed and a photoresistor that detects when the candy has fallen into the holder. Once the candy has fallen into the holder Nigel will spin around 180 degrees and offer the candy. He will then wait to see if the candy is taken. If the candy is taken, Nigel will rejoice and dance and then track another person, if however the candy is not taken, Nigel will turn around and throw the candy at the person and then go into hidemode.

Hide mode simply consists of Nigel searching for a dark place to hide. He will roam around the room leaning towards shadows searching for the darkest spot. When he has found it he will wait there "hiding" for about 10 seconds, then after he will return into tracking mode. This method of hiding is achieved by a set of three photoresistors placed on the top of the platform that indicate where Nigel should turn and where the darkest spot is.

This system behaviors is the outline of what Nigel is designed to do.

Introduction

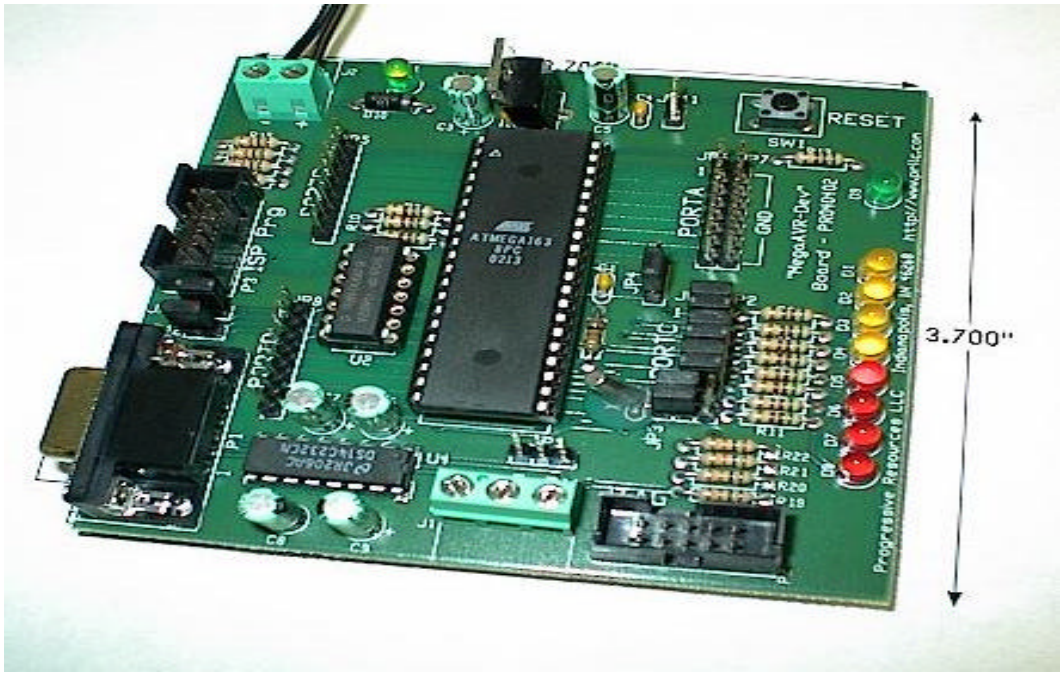
The story of Nigel begins with an initial idea of a robot that could go around a room, interact with people by asking a few questions and based on the answers received via voice recognition, Nigel would then be obliged to take that persons picture. I felt that this idea might prove to be rather difficult being that this is my first robot, thus, spawning from the original idea, Nigel: The Autonomous Candy Man was born. I felt that if I were to build any type of robot, I would like it to resemble some interesting interactive qualities, such as adopting certain behavioral traits. So, in essence I want to try to give him some type of personality, hence setting my goal to create a robot that will exhibit funny, yet, child-like behavior.

Nigel's process is as follows: he will initially travel around aimlessly in search of a person that attracts him, like someone with the color blue. Once he is attracted he will approach the person, stop and offer candy. Nigel is happy if the person is obliged and takes the candy, but if the candy is not taken Nigel will become frustrated and will then throw the candy and run away in search of a dark spot to hide.

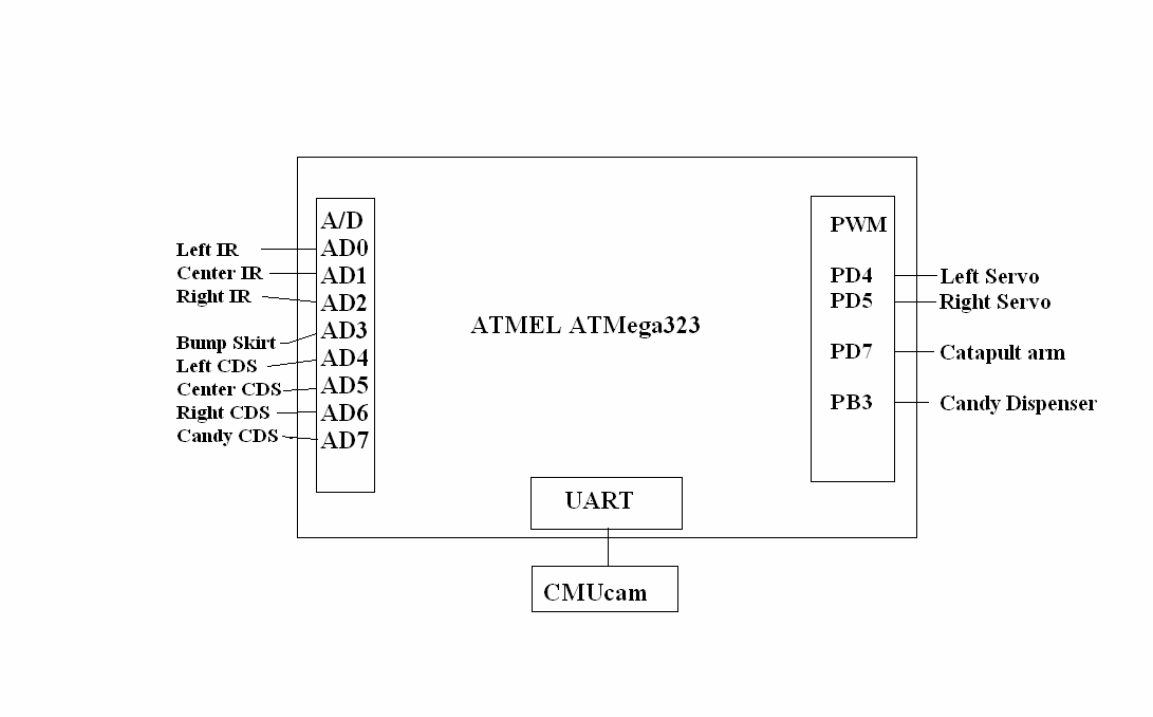
Nigel's brain consists of the Atmel AVR-Mega323. This will control mostly all of Nigel's functions, namely his servos, sensors and the CMUcam that will be used to find a person to offer candy to.

Integrated System

The board being used is the Atmel AVR-Mega323; I will use this board to drive all of Nigel's systems, namely 3 IR Rangers, a bump skirt, CMUcam, servos, and CdS cells. Shown below is the AVR-Mega323.



The following diagram is a description of Nigel's
System:



Nigel's functions consist of:

- Object Avoidance
- Dispenser system and servo wheels
- Color tracking
- Searching for dark areas

Nigel's object avoidance system consists of using 3 Sharp GP2D12 IR rangers and 4 bump switches that comprise its bump skirt. The first four A/D pins on the board receive analog values from this feature.

The dispenser system designed for Nigel consists of 2 servos and a Cds cell. Two servo wheels control Nigel's movements across the floor. All of the servos are controlled by the four PWM modes, and the CdS cell's analog values is receive in A/D pin 7

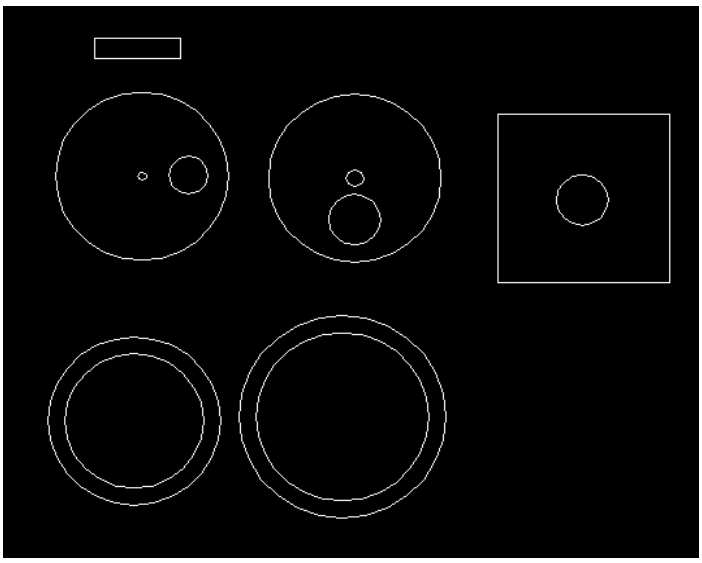
A CMUcam that communicates using the boards USART system achieves color tracking.

When Nigel is looking for a place to hide, 3 CdS cells are used in which A/D pins 4:6 receive the cells analog values.

Mobile Platform

The planning of my mobile platform consists of a simple circular base about 9” in diameter with two wheels and a castor. I implemented a bridge that is a 3.5” radius half circle that sits 3” above the base that will enable mounting for my CMUcam and my photoresistors. The Development board is mounted under the bridge, and the battery packs placed on the outside of the bridges legs. A bump skirt is also placed around the perimeter of the base.

The arm used to serve and throw the candy works in a catapult type fashion. An 8” wooden stick that is .25” thick is attached to the right side of the base and placed in a spring loaded shaft that is attached to the back of the base. The dispenser was designed out of wood that consisted of a circular base, 2.5” in diameter, with a hole about .75” in diameter, two .25” thick spacing rings each 2.5” in diameter and a circular top with the same diameter and a smaller hole about the diameter of a Skittle. To mount the candy holder a square platform was built with a hole the size of the bottle mouth and attached to the dispenser ring by an outer ring connected to it. The AutoCad design and the dispenser/catapult placement are shown below.



As seen by the picture, there is a servo that is placed by the catapult with wooden reinforcement attached to it to enable a it to pull the stick down, compressing the spring, and then releasing it causing the stick to fly forward thus throwing the candy. The ball used to hold the candy is and old "smash ball" lined with a piece of cut up index card to enable the candy to slide. I picked this ball because of its ability to absorb shock and will able the candy to fall into it without it bouncing out. I also cut a hole in the bottom of the ball for a CdS cell. The idea, is that when the candy falls into the ball it will land on top of the hole covering the photoresistor, this will allow Nigel to know when the candy is dispensed or not. The slide that the candy falls from was simply made from cardboard. I had originally wanted to use rubber tubing but I found later the candy would get stuck on the way down and would not slid very well. Hence, making cardboard an easy solution.

The placement of the IR rangers was specifically placed across the front of the bottom base spaced approximately 30 degrees apart. I found that this was a good place to put them because of there wide range of IR emission that they would over lap one another, thus spanning across the entire front area of the platform. A picture of the IR placement is shown below:

Nigel's platform was a very important part of Nigel's success. I spent a very long time planning and placing the parts and putting them together. Proper distribution of weight was a key factor. Sometimes when reversing Nigel would "nose dive" as all the weight falls forward at an abrupt reverse movement. I counteracted that by placing the battery packs closer to the rear of the platform so Nigel would be excessively back heavy so when the spring action catapult flies forward it wont knock Nigel forward with it.

Actuation

Nigel will roam about on two wheels and a caster. The wheels used are R170-Blue-Black wheels purchased at Acroname. Each wheel is connected to a parallax R174 continuous servo also purchased at Acroname. The Candy dispenser also uses the parallax R174 continuous servo. For the lever that actuates the catapult, I originally tried to use the the parallax cont. servo but found that it did not have enough torque to pull the lever back against the compression of the spring. Being that it only exerted a torque of 44oz, I invested in a high torque servo from Mark III Robot store that exerted 69oz of torque. This servo had to be hacked being that I couldn't find a continuous one. Upon actuation the high torque servo was able to pull the lever back with ease.

Each servo has a neutral pulse width of 1500us. Which translates to 666.7hz. The easiest way to actuate the servos is to utilize the PWM on the board. The Atmel323 has four PWM modes, so for each output compare register, a value greater then the

neutral would cause the servo to spin clockwise and a value less then the neutral cause the servo to spin clockswise.

Other servos can be purchased but they are not continuous, thus having to be hacked, one can obtain these at Acroname for a cheaper price. I chose to go with the continuous ones because they are better quality where the plastic bushings in the regular servos are replaced with ball bearings, and the potentiometer doesn't have to be calibrated. As a precaution, some of the other robots using the standard servo had problems with them burning out. I have not had any trouble with the parallax servos so far.

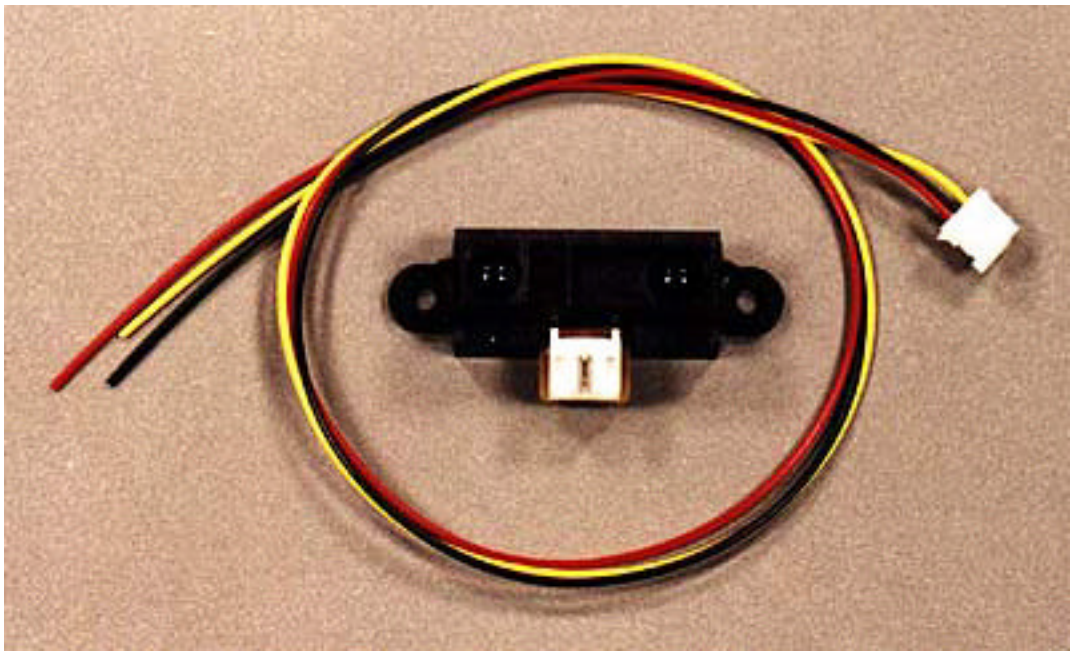
Sensors

All of Nigel's sensors are connected thru the A/D with exception to the CMUcam. Nigels sensors are as follows:

- Three Sharp GP2D12 IR Rangers
- Four CdS "photoresistors"
- CMUcam

IR Rangers

Nigel uses three Sharp GP2D12 IR rangers spaced across the front of the platform. These rangers detect objects between 4" – 30". For Nigel's purposes this range was adequate. Interfacing the IR's was very simple. Each IR uses a +5V power, Gnd, and an analog out, Vo. By using the A/D converter on the board I was able to convert the analog output signal to a digital signal that was displayed on the board's leds. Values were documented based on the distance of the object and from those values I determined at what point should Nigel move to avoid the object. A list of obtained digital values versus distance is shown in the Experimental Data section. After much trial and error, it was best to have Nigel turn when an object was within 6", thus once the digital value was greater then or equal to 5, Nigel avoided the object. An illustration of the ranger is shown below



The only problem using the IR's is that they will not detect anything less then 4", so if Nigel "saw" something and turned towards an object that was less than 4" away there was no way for Nigel to see it. Thins rods, like the leg of a chair, also posed a problem because the transmitted signal might not have reflected back every time so the ranger would get inaccurate readings. Nigel would sometimes avoid it successfully but other times he would move for a second then go into the chair leg. I feel that these rangers are very accurate and work very well with the exception of a few scenarios.

Bump Skirt

In case the IR fail I have equipped Nigel with a bump skirt that will allow him to know if he has bumped into something. Basically, I have set up a series of switches that surround the base of the platform, three around the front right, middle and left, and directly in the back. These switches serve as detectors where if one is pressed it sends a certain voltage to an A/D pin telling the board that Nigel has hit something. This bump network can be constructed by using a simple voltage divider circuit. Each bump switch is connected to 5V and to a corresponding resistance so the analog signal will be a different value depending on which switch is set. This will allow Nigel to know which side was hit. The voltage divider circuit is displayed in the Experimental Data section.

CdS Photoresistors

I will be using 3 CdS cells or photoresistors in which Nigel uses when he seeks a place to hide. Each photoresistor is enclosed in a dark tubing so to control incident light from random area. Each CdS cell gives an analog reading depending on the amount of light the resistor receives. The more light the higher the change in the resistor value thus returning an higher analog signal. To find a dark area Nigel will turn in the direction of the photoresistor that sends the lowest value.

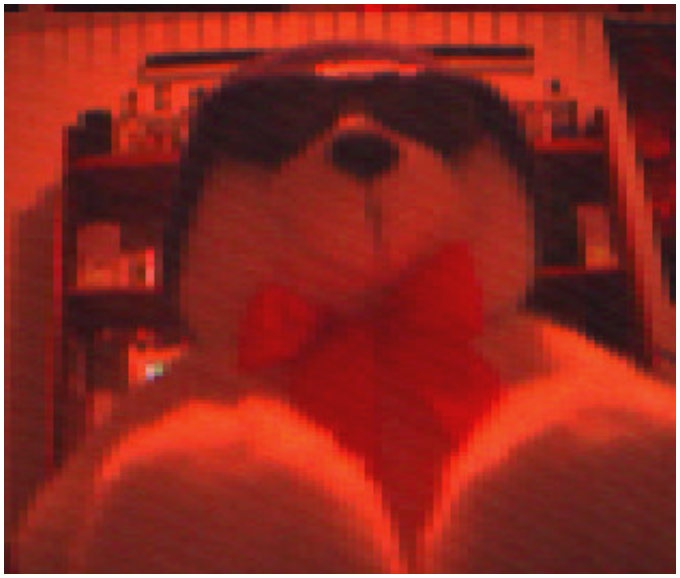
Nigel will also use a CdS cell to determine if the candy in the candy holder has been taken or thrown. Rather then using a break beam, the photresistor will be set under the candy so that if candy is present, there will be no light transmitted onto the resistor. Once the candy is taken, the resistor will get flooded with light and then go and track another person to offer candy to.

Since lighting is never consistent, it was necessary to add a self-calibration function so that wherever I take Nigel, it will self adjust to the lighting conditions and set its own light threshold. The constant threshold for the candy photoresistor is always a little less than the value of light that initially is first seen. Nigel will offer the candy after it knows it had received the candy in the ball, thus the ball's photoresistor value is less then the threshold value. This is because, when the A/D converts analog to digital an x amount of times the digital value changes slightly so it is never the same. Under extreme lighting conditions such as fluorescent, the value can jump between A0-B0, thus it is ideal to initialize its threshold values a certain number less then what it receives. I found that this system worked rather well in dim areas but the brighter it got, the more inconsistent the balls photoresistor was because Nigel would think it got the candy before it actually did. I fixed the problem by finding what the photoresistor value was under fluorescent lighting, and then subtracting $1C in hex from it and making that my threshold. That seemed to make the calibration a lot more accurate.

CMUcam

The CMUcam's abilities include, position and size tracking of colorful or bright objects, measure the RGB and YUV characteristics of an image region, display images and bitmap of objects via serial port. For Nigel's purpose, the use of the CMUcam is

quite simple, Nigel will utilize the color tracking feature and interface with the board to control its servos to approach a person wearing the color blue. The images "dumped" by the camera had a very dark red tint thus discoloring objects placed in front of it. A picture of a stuffed bear displaying a variety of colors is shown below to demonstrate the images on the cam.



The bear is white, with a red bowtie around its neck, black glasses and a blue hat. This picture was taken under bright lighting conditions in my room. As you can see, this type of lighting is not good and tracking a color like blue would prove to be difficult. The best images were obtained when I was under fluorescent lighting, colors were more distinct and the picture was much clearer.

Camera Integration

The CMUcam itself uses a SX28 microcontroller that communicates either through a RS-232 or a TTL serial port. To integrate with my AVR-MEGA progressive development board I communicated through the RS-232 and level shifted serial port. Communication through this port uses the following parameters:

- 115,200 baud
- 8 data bits
- 1 Stop bit
- No Parity

- No Flow control

These parameters are ideal for communication with a computer, i.e. hyper terminal. However, one can interface it with a board just the same. The camera has several commands that are rather easy to use; the command that was relevant for Nigel was the track color command or "TC". By using the CMUcam software that came with the cam, I was able to easily dump a frame, get color coordinates and obtain values for the minimum and maximum intensities of the camera's main colors red, blue, and green. Once those values are entered, the track color function can be assessed and the once the colored object is placed in front of the cam the green light will blink indicating that the color being tracked is found, what is being returned from the camera is an M-type packet which consists of:

M mx my x1 y1 x2 y2 pixels confidence

This string is what I will use to calibrate Nigel to chase the color blue.

M=represents M type packet

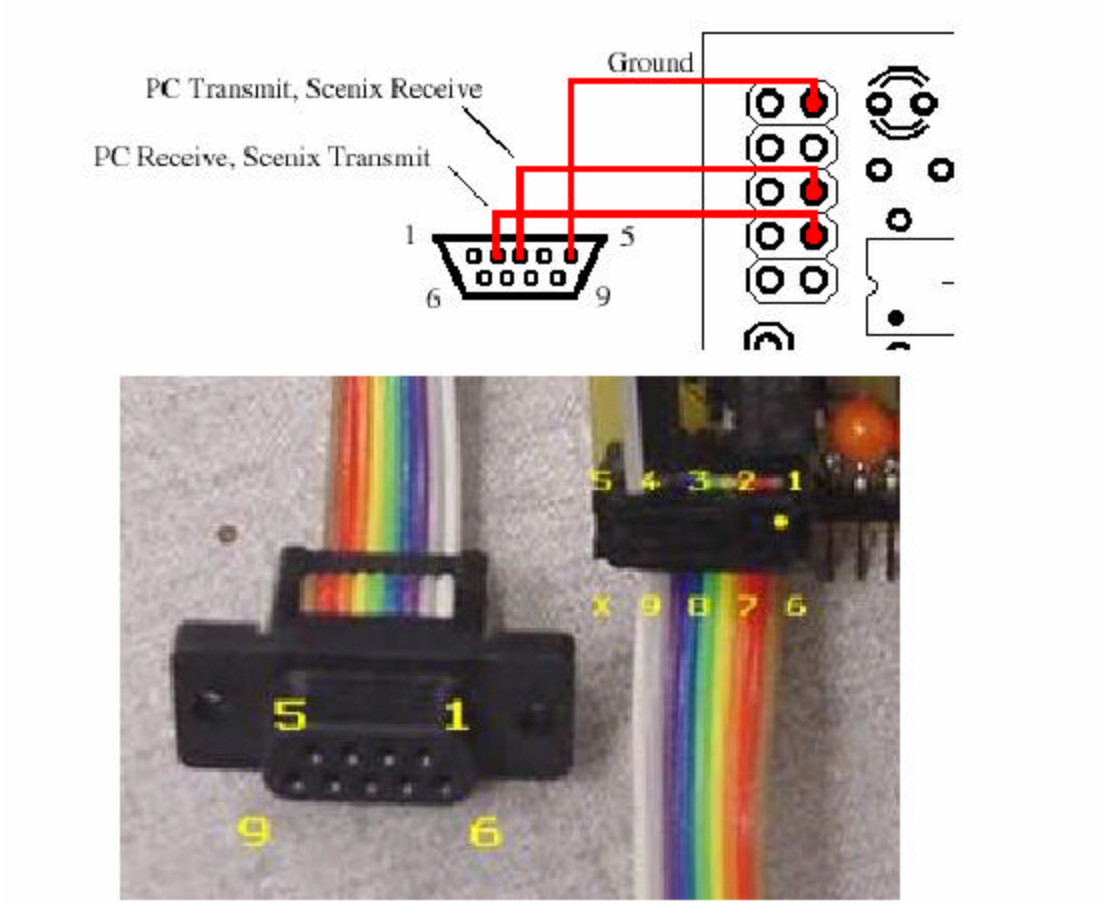
mx=middle mass x

my=middle mass y

x1, y1, x2, y2 represent the coordinates of the box that the camera sees when it is tracking.

For Nigels purposes, I am only concerned with the middle mass x (mx) where he will turn left, right or go straight depending on where the color goes.

This method of obtaining "min and max" RGB color coordinates was extremely easy, but communicating with the board proved to be rather tedious. The problem was that I was able to send commands to the camera but not able to receive packets of info back. After much frustration and collaboration, a method of communication was established. Before communication between the cam and the board can be established it is necessary to figure out where the Tx, Rx pins are located on the cam. As shown below:



To receive packets from the camera, the Tx and Rx pins on the cam must be crossed to connect properly to the Rx and Tx pins of the board. In other words, pins 2 and 3 from the camera need to be crossed so the transfer (pin2) of the cam is the receive pine of the board (pin 3) and the receive pin of the cam (pin3) is connected to the transfer of the board (pin 2). By doing this and cutting all the other wires we were able to establish a clean and steady communication system. Thus, I was able to send a string of commands and receive the middle x packet byte and display it on PortC.

Behaviors

Nigel will exhibit 3 different types of behaviors:

- Tracking Behavior/Tracking mode
- Dispensing and offering Behavior/Dispensing mode
- Hiding behavior/Run and Hide Mode

Tracking Mode

In this mode Nigel is using the CMUcam to track people where the color blue. He will spin twice in search of the blue color, if nothing has been tracked Nigel will then go into avoidance mode where he will wander off randomly for a couple of seconds utilizing the object avoidance feature, thus moving to another place to start tracking .

Dispensing Mode

When Nigel has found a person to approach, Nigel will come up to the person and get within a few inches from there feet, stop, backup and then enable the candy dispensor to spin until the candy holder has received the candy, at that moment, Nigel will turn 180 degrees and hold the candy out in front of the person. If the candy is taken Nigel will rejoice and do a little dance, the length of the dance depends on how long the person takes to take the candy, meaning Nigel is getting unhappy the longer one waits. If the candy is not taken, Nigel will turn around and throw the candy at the person, thus leading into its next behavior.

Run and Hide Mode

This mode utilizes the photoresistors to find a dark place to hide. After the candy is thrown Nigel immediately goes into hide mode, and will not stop until it has found a dark place to hide. After finding a dark place, Nigel will hang out there for approximately 10 seconds then come out and go into tracking mode again.

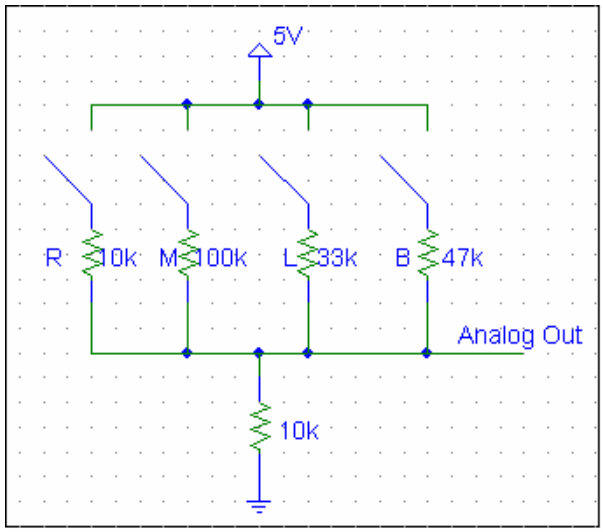
Experimental Layout and Results

The first experiment was to get values for my IR rangers and get digital values that correspond to distance. By using a simple A/D conversion code and displaying it on to PortC, I was able to obtain the following relation. Because the lower nibble of the 8 bit digital values displayed seemed to change extremely rapidly, the documented values only correspond to the upper nibble. The values on the leds are displayed active low.

Distance(inches)	Digital value
4	7
5	6
6	5
7	4
8	4
9	3
10	3
11	3
12	2

After much trial and error it was best to have Nigel turn away from an object at 6” or when the upper nibble received a value greater than or equal to $50.

The next experiment tested the bump skirt. I used the following voltage divider to get different analog value for each switch.

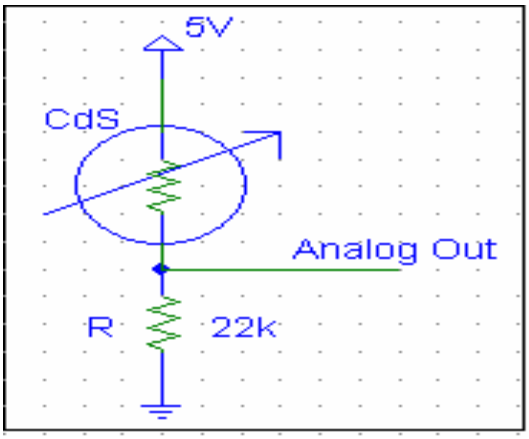


Values from each switch are as follows:

Switch Pushed	Digital Value(hex)
None	$00
Back	$0C
Left	$10
Middle	$06
Right	$30
Left and Back	$1E
Middle and Back	$13
Middle and Left	$18
Right and Back	$3D
Right and Left	$3F
Right, Left, and Back	$40
Right and Middle	$38
Right, Middle, Back	$3F
Right, Middle, Left	$44
All switches hit	$4C

Of course not all of these situation are realistic or bound to happen often so I threw all the values that combined with the back switch to help simplify the code. My reasoning is that since Nigel is a forward moving robot, there shouldn't be very many times when the back is hit along with any of the other switches, hence, only the value when the back is hit is used.

For the photoresistors, I used a voltage divider again, but this time I use it do get an analog voltage out of each individual CdS cell. The divider is as follows:



By using this circuit I was able to get accurate readings for each resistor. By taking readings in a well light area for a dark area I was able to compare the value for each photo resistor.

	Right Photo	Left Photo	Center Photo	Candy Photo
Bright	$A0	$E0	$EB	$C0
Dim	$20	$36	$34	$30

From these extremes I was able to set a relatively good threshold for when Nigel hides and also when the candy has been dispensed or taken.

Conclusion

Nigel turned out to be a complete success. I was able to accomplish almost all that I had set out to do. I only wish that I had more time to add a voice chip so Nigel can say something to people before he offers the candy. I think that after using the CMUcam, I feel that it might have been better to have used a pyroelectric sensor in which Nigel could approach anybody rather then be limited to just color. I did purchase one and will later implement that in Nigel and see how it works. I would also change the platform in a way in which connecting to the board was easier. I plan on making another robot using the tools I gained in this class and hopefully learn and gain invaluable experience from it.

Acknowledgements

I would like to acknowledge the following for all their help:

Dr. Arroyo – help on ideas for the Robot

Dr. Schwartz – Critiquing my robot to motivate me to make it better

Tas: Uriel Rodriquez and Jason Plew – suggestions and knowledge that helped make Nigel a success

Kyle Tripician – "CMU man" for figuring out how the heck that thing worked

Steve Van der Ploeg – Voice Chip

The Spring 2003 IMDL classmates who I collaborated with.

Parts and Prices

3x R174 Parallax cont. rotation servos	@13.00 ea	www.acroname.com
2x R170 Blue-Black wheel	@3.50 ea	www.acroname.com
1x GWS S03TFX 2BB High torque servo	@10.50 ea	
http://www.junun.org/MarkIII/Store.jsp#item19		
3x Sharp GP2D12 IR Rangers	@13.50 ea	www.acroname.com
4x CdS "Cadmium Sulfide" cells	Free from lab	
1x CMUcam	@109.00 ea	www.seattlerobotics.com
1xAtmega323 board W/chip	@56.00 ea	http://www.prllc.com

Appendix

This Appendix contains Nigel's Code along with experimental code that was used to calibrate and test Nigel's parts.

```
/*Brian Ruck
Code to test all PWM modes
*/
#include <io.h>
#include <sig-avr.h>
#define AVR_MEGA 0
int main(void)
{
outp(0xFF,DDRD);
outp(0xFF,DDRB);
outp(0x64,TCCR0);    //enable PWM in 8bit timer/counter control register0
outp(0x66,TCCR2);    //enable PWM in 8bit timer/counter control register2
outp(0xA1,TCCR1A);//enable PWM in 16 timer/counter control register1A
outp(0x04,TCCR1B); //prescale 16bit timer ck/256
outp(0x00,TCNT0);
outp(0x00,TCNT2);
outp(0x00,TCNT1H);
outp(0x00,TCNT1L);
outp(0x00,OCR1AH);
outp(0x16,OCR1AL); //PORTD pin 5 Right Servo
outp(0x00,OCR1BH);
outp(0x12,OCR1BL); //PORTD pin 4 Left Servo
outp(0x12,OCR0);     //PORTB pin 3 servo on candy dispensor
outp(0x12,OCR2);     //PORTD pin 7 servo on arm
for (;;) {}
}
```

****

```
/*A/D test code
value output on PortC*/
#include <io.h>
#include <sig-avr.h>
#define AVR_MEGA 0
typedef unsigned char  u08;
int main( void ){
        u08 led,i,j,k;
        outp(0xff,DDRC);            /* use all pins on Portc for output */
        outp(0x80,ADCSR);           /* turn A/D on */
        outp(0x27,ADMUX);           //left adjusted, channel 0
        while(1){
                sbi(ADCSR,ADSC);            /* start conversion */
                loop_until_bit_is_clear(ADCSR,ADSC);/* Checking if done */
                led=inp(ADCH);                      /*getting value */
                outp(led,PORTC);
                for(i=0; i<255; i++);
                for(j=0; j<255; j++);
                for(k=0; k<255; k++);
                }
}
```

```
/*
Brian Ruck
Tracking code
using
Kyle Tripician's
UART code.
*/
#include <io.h>
#include <sig-avr.h>
#include <stdlib.h>
#include <interrupt.h>
#include <progmem.h>
typedef unsigned char u08;
typedef unsigned int u8;
typedef          char s08;
typedef unsigned short u16;
typedef          short s16;
#include "uart.h"
#include "delay.h"
volatile u8 temp;
volatile u8 i=0;
volatile u8 cmudat[9];
u08 rmotor=1;
u08 lmotor=2;
u08 dispenser=3;
u08 flingage=4;
int fwd=100;
int turn=50;
int rev=-100;
int stop=0;
int dispense=60;
```

```
int fling=70;
int b,c,x,y,z;
SIGNAL(SIG_UART_RECV){
        temp=inp(UDR);
        if(temp != 0x3A){
                if(temp == 0x20 || i==9){
                i=0;
                }
                else if(i==0){
                        if(temp == 0xFF){
                                cmudat[i]=temp;
                                i++;
                        }
                }
                else if(temp !=0x20 && i<9){
                cmudat[i]=temp;
                i++;
                }
        }
}
void servo(s,d){
        if(s==lmotor){
                if(d==fwd){
                        outp(0x00,OCR1BH);
                        outp(0x23,OCR1BL); //left motor fastest forward//
                        }
                else if(d==rev){
                        outp(0x00,OCR1BH);//left motor reverse//
                        outp(0x0A,OCR1BL);
                        }
                else if(d==turn){
```

```
                outp(0x00,OCR1BH);//left motor slows to enable left turn//
                outp(0x14,OCR1BL);
                }
            else if(d==stop){
                outp(0x00,OCR1BH);//left motor stops//
                outp(0x12,OCR1BL);
                }
            }
        else if(s==rmotor){
            if(d==fwd){
                outp(0x00,OCR1AH);
                outp(0x0A,OCR1AL);          //right motor fastest forward//
                }
            else if(d==rev){
                outp(0x00,OCR1AH);//right motor reverse//
                outp(0x23,OCR1AL);
                }
            else if(d==turn){
                outp(0x00,OCR1AH);//right motor slows to enable left
turn//
                outp(0x10,OCR1AL);
                }
            else if(d==stop){
                outp(0x00,OCR1AH);//right motor stops//
                outp(0x12,OCR1AL);
                }
            }
        else if(s==dispenser){
            if(d==dispense){
                outp(0x0F,OCR0);     //dispense candy
                }
```

```
                else if(d==stop){
                        outp(0x12,OCR0);     //candy dispensor is stopped
                        }
                }
        else if(s==flingage){
                if(d==fling){
                        outp(0x16,OCR2);     //through candy
                        }
                else if(d==stop){
                        outp(0x12,OCR2);     //stop servo to hold arm down
                        }
                }
        }
void blink(void){
        uartstring("L1 1\r");
        delay(1500);
        delay(1500);
        delay(15000);
        uartstring("L1 0\r");
        delay(15000);
        delay(1500);
        delay(1500);
        }
int main(void){
outp(0xFF,DDRD);    //PORTD set output signal for servos left(PD4), right(PD5),
flingage(PD7)
outp(0xFF,DDRB);    //PORTB set output signal for dispensor(PB3)
outp(0x64,TCCR0);    //enable PWM in 8bit timer/counter control register0
outp(0x66,TCCR2);    //enable PWM in 8bit timer/counter control register2
outp(0xA1,TCCR1A);          //8 bit PWM//
outp(0x04,TCCR1B);          //prescale clock by 256//
```

```
outp(0xFF,DDRC);
delay(10000);
uartinit();
delay(10000);
uartstring("PM 1\r");
delay(1000);
uartstring("RM 3\r");
delay(1000);
sei();
delay(1000);

while(1){
uartstring("TC 42 52 33 53 40 90\r");
outp(cmudat[2],PORTC);
delay(10000);
        if(cmudat[2]==0x00){                    //tracking color to chase
                servo(lmotor,fwd);
                servo(rmotor,rev);
                delay(9000);

                servo(lmotor,stop);
                servo(rmotor,stop);
                delay(40000);
        }
        if(cmudat[2]>0x00 && cmudat[2]<0x19){   //track to left
                servo(lmotor,turn);
                servo(rmotor,fwd);
                for(x=0;x<255;x++){
                for(y=0;y<255;y++){
                for(z=0;z<100;z++){
                }
```

```
                    }
                    }
                }
                if(cmudat[2]>=0x19 && cmudat[2]<=0x38){          //track center
                    servo(lmotor,fwd);
                    servo(rmotor,fwd);
                    for(x=0;x<255;x++){
                    for(y=0;y<255;y++){
                    }
                    }
                }
                if(cmudat[2]>0x38){                              //track to right
                    servo(lmotor,fwd);
                    servo(rmotor,turn);
                    for(x=0;x<255;x++){
                    for(y=0;y<255;y++){
                    for(z=0;z<100;z++){
                    }
                    }
                    }
                }
        //      blink();
                }
}
```

```
//Nigel's code
//Brian Ruck
#include <io.h>
#include <sig-avr.h>
#include <stdlib.h>
#include <interrupt.h>
#include <progmem.h>
typedef unsigned char u08;
typedef unsigned short u16;
typedef unsigned int u8;
typedef        char s08;
typedef        short s16;
u08 lir;
u08 cir;
u08 rir;
u08 bump;
u08 rphoto;
u08 cphoto;
u08 lphoto;
u08 candyphoto;
u08 darkcandy/*=0x8F*/;
u08 rmotor=1;
u08 lmotor=2;
u08 dispenser=3;
u08 flingage=4;
u08 distc=0x40;
u08 distf=0x50;
u08 distrlf=0x70;
u08 darkl/*=0x9F*/;
u08 darkr/*=0x6F*/;
u08 darkrr/*=0x5F*/;
```

```
u08 darkhole=0x36;
int fwd=100;
int turn=50;
int rev=-100;
int srev=-50;
int stop=0;
int dispense=70;
int fling=80;
int hidemode=0;
int dispensemode=0;
int taken=0;
int track=0;
u08 time=0x2C;
int s,d,x,y,z,a,b,c,e,j,k,v,l;
#include "uart.h"
#include "delay.h"
volatile u8 temp;
volatile u8 i=0;
volatile u8 cmudat[9];
u08 adval[8];                                   //value recieved in specific ad
channel
u08 adchan[8];

SIGNAL(SIG_UART_RECV){
        temp=inp(UDR);
        if(temp != 0x3A){
                if(temp == 0x20 || i==9){
                i=0;
                }
                else if(i==0){
                        if(temp == 0xFF){
```

```
                    cmudat[i]=temp;
                    i++;
                }
            }
            else if(temp !=0x20 && i<9){
            cmudat[i]=temp;
            i++;
            }
        }
}
void servo(s,d){
        if(s==lmotor){
            if(d==fwd){
                outp(0x00,OCR1BH);
                outp(0x23,OCR1BL); //left motor fastest forward//
                }
            else if(d==rev){
                outp(0x00,OCR1BH);//left motor reverse//
                outp(0x0A,OCR1BL);
                }
            else if(d==turn){
                outp(0x00,OCR1BH);//left motor slows to enable left turn//
                outp(0x14,OCR1BL);
                }
            else if(d==stop){
                outp(0x00,OCR1BH);//left motor stops//
                outp(0x12,OCR1BL);
                }
            else if(d==srev){
                outp(0x00,OCR1BH);
                outp(0x10,OCR1BL);
```

```
                }
        }
    else if(s==rmotor){
        if(d==fwd){
                outp(0x00,OCR1AH);
                outp(0x0A,OCR1AL);          //right motor fastest forward//
                }
        else if(d==rev){
                outp(0x00,OCR1AH);//right motor reverse//
                outp(0x23,OCR1AL);
                }
        else if(d==turn){
                outp(0x00,OCR1AH);//right motor slows to enable left
turn//
                outp(0x10,OCR1AL);
                }
        else if(d==stop){
                outp(0x00,OCR1AH);//right motor stops//
                outp(0x12,OCR1AL);
                }
        else if(d==srev){
                outp(0x00,OCR1AH);
                outp(0x14,OCR1AL);
                }
        }
    else if(s==dispenser){
        if(d==dispense){
                outp(0x0C,OCR0);     //dispense candy
                }
        else if(d==stop){
                outp(0x12,OCR0);     //candy dispenser is stopped
```

```
                }
            }
        else if(s==flingage){
            if(d==fling){
                outp(0x16,OCR2);    //through candy
                }
            else if(d==stop){
                outp(0x12,OCR2);    //stop servo to hold arm down
                }
            }
        }
    void conv(void){
        c=0x00;                                 //intialize rangers for
conversion//
        adchan[0]=0x20;                         //channel 0 left ir//
        adchan[1]=0x21;                         //channel 1 center ir//
        adchan[2]=0x22;                         //channel 2 right ir//
        adchan[3]=0x23;                         //channel 3 bump switches//
        adchan[4]=0x24;                         //channel 4 left
photoresistor//
        adchan[5]=0x25;                         //channel 5 center
photoresistor//
        adchan[6]=0x26;                         //channel 6 right
photoresistor//
        adchan[7]=0x27;                         //channel 7 candy
photoresistor//

        for(b=1;b<=8;b++){
            outp(adchan[c],ADMUX);
            sbi(ADCSR,ADSC);                    // start conversion
            loop_until_bit_is_clear(ADCSR,ADSC);    // checking if done
```

```
                adval[c]=inp(ADCH);                 //place value in adval array//
                c++;
                }
        lir=adval[0];                   //store val of left ir//
        cir=adval[1];                   //store val of center ir//
        rir=adval[2];                   //store val of right ir//
        bump=adval[3];                      //store val of bumpswitch//
        lphoto=adval[4];                //store val of left photoresistor//
        cphoto=adval[5];                //store val of center photoresistor//
        rphoto=adval[6];                //store val of right photoresistor//
        candyphoto=adval[7];            //store val of candy photoresistor//
        outp(cphoto,PORTC);
}

void calibrate(){                       //self calibration function
        conv();
        darkl=lphoto-0x60;
        darkr=rphoto-0x60;
        darkcandy=candyphoto-0x15;
        }

void dispensecandy(void){
                servo(lmotor,turn);                         //begin
dispense mode
                servo(rmotor,turn);
                servo(dispenser,stop);
                servo(flingage,stop);
                for(x=0;x<255;x++){
                for(y=0;y<255;y++){
                for(z=0;z<255;z++){
                }
```

```
                    }
                    }
                    servo(lmotor,stop);
                    servo(rmotor,stop);
                    servo(dispenser,stop);
                    servo(flingage,stop);
                    for(x=0;x<255;x++){
                    for(y=0;y<255;y++){
                    for(z=0;z<255;z++){
                    }
                    }
                    }
                    servo(lmotor,rev);
                    servo(rmotor,rev);
                    servo(dispenser,stop);
                    servo(flingage,stop);
                    for(x=0;x<255;x++){
                    for(y=0;y<255;y++){
                    for(z=0;z<255;z++){
                    }
                    }
                    }
                    conv();
                    while(candyphoto>=darkcandy){
//dispense candy function
                        servo(lmotor,stop);
                        servo(rmotor,stop);
                        servo(dispenser,dispense);
                        servo(flingage,stop);
                        conv();
                    }
```

```
servo(lmotor,stop);
servo(rmotor,stop);
servo(dispenser,stop);
servo(flingage,stop);
for(x=0;x<255;x++){
for(y=0;y<255;y++){
}
}
servo(lmotor,stop);
servo(rmotor,stop);
servo(dispenser,stop);
servo(flingage,stop);
for(x=0;x<255;x++){
for(y=0;y<255;y++){
}
}
servo(lmotor,rev);
servo(rmotor,fwd);
servo(dispenser,stop);
servo(flingage,stop);
for(x=0;x<255;x++){
for(y=0;y<255;y++){
for(z=0;z<500;z++){
}
}
}
servo(lmotor,stop);
servo(rmotor,stop);
servo(dispenser,stop);
servo(flingage,stop);
for(v=0;v<11;v++){
```

```
for(l=0;l<5000;l++){
conv();
        if(candyphoto>=darkcandy){
                servo(lmotor,rev);
                servo(rmotor,fwd);
                servo(dispenser,stop);
                servo(flingage,stop);
                for(x=0;x<255;x++){
                for(y=0;y<255;y++){
                for(z=0;z<500;z++){
                }
                }
                }
                servo(lmotor,fwd);
                servo(rmotor,fwd);
                servo(dispenser,stop);
                servo(flingage,stop);
                for(x=0;x<255;x++){
                for(y=0;y<255;y++){
                for(z=0;z<255;z++){
                }
                }
                }

        taken=1;
        l=5000;
        }
}
}
conv();
        if(candyphoto<darkcandy){
```

```
servo(lmotor,fwd);
servo(rmotor,rev);
servo(dispenser,stop);
servo(flingage,stop);
for(x=0;x<255;x++){
for(y=0;y<255;y++){
for(z=0;z<500;z++){
}
}
}
servo(lmotor,stop);
servo(rmotor,stop);
servo(dispenser,stop);
servo(flingage,stop);
for(x=0;x<255;x++){
for(y=0;y<255;y++){
for(z=0;z<255;z++){
}
}
}
servo(lmotor,stop);
servo(rmotor,stop);
servo(dispenser,stop);
servo(flingage,fling);
for(x=0;x<255;x++){
for(y=0;y<265;y++){
for(z=0;z<400;z++){
}
}
}
servo(lmotor,fwd);
```

```
                    servo(rmotor,rev);
                    servo(dispenser,stop);
                    servo(flingage,stop);
                    for(x=0;x<255;x++){
                    for(y=0;y<255;y++){
                    for(z=0;z<500;z++){
                    }
                    }
                    }
                hidemode=1;
                }
            }                                   //end dispensemode


void bumpswitch(void){
    if (bump>=0x0C && bump<=0x11){              //back switch hit
        servo(lmotor, fwd);
        servo(rmotor, fwd);
        for(x=0;x<255;x++){
        for(y=0;y<255;y++){
        }
        }
    }
    if(bump>0x11 && bump<=0x12){                //left switch hit
        servo(lmotor,rev);
        servo(rmotor,rev);
        for(x=0;x<255;x++){
        for(y=0;y<255;y++){
        for(z=0;z<600;z++){
        }
```

```
        }
        }
        servo(lmotor,fwd);
        servo(rmotor,turn);
        for(x=0;x<255;x++){
        for(y=0;y<255;y++){
        for(z=0;z<255;z++){
        }
        }
        }
}
else if(bump>=0x06 && bump<=0x08){          //middle switch hit
        servo(lmotor,rev);
        servo(rmotor,rev);
        for(x=0;x<255;x++){
        for(y=0;y<255;y++){
        for(z=0;z<600;z++){
        }
        }
        }
        servo(lmotor,rev);
        servo(rmotor,fwd);
        for(x=0;x<255;x++){
        for(y=0;y<255;y++){
        for(z=0;z<255;z++){
        }
        }
        }
}
else if(bump>=0x30 && bump<=0x34){          //right switch hit
        servo(lmotor,rev);
```

```
                servo(rmotor,rev);
                for(x=0;x<255;x++){
                for(y=0;y<255;y++){
                for(z=0;z<600;z++){
                }
                }
                }
                servo(lmotor,turn);
                servo(rmotor,fwd);
                for(x=0;x<255;x++){
                for(y=0;y<255;y++){
                for(z=0;z<255;z++){
                }
                }
                }
        }
}
void avoidance(){
                conv();
                if(bump>0x01){
                        bumpswitch();
                }
                else{
                if(lir<distf && cir<distf && rir<distf){
                        servo(lmotor,fwd);
                        servo(rmotor,fwd);
                        for(x=0;x<255;x++){
                        for(y=0;y<255;y++){
                        }
                        }
                }
```

```
if(lir<distf && cir<distf && rir>=distf){
        servo(lmotor,turn);
        servo(rmotor,fwd);
        for(x=0;x<255;x++){
        for(y=0;y<255;y++){
        for(z=0;z<100;z++){
        }
        }
        }
}
if(lir<distf && cir>=distf && rir<distf){
        servo(lmotor,rev);
        servo(rmotor,fwd);
        for(x=0;x<255;x++){
        for(y=0;y<255;y++){
        for(z=0;z<500;z++){
        }
        }
        }
}
if(lir<distf && cir>=distf && rir>=distf){
        servo(lmotor,stop);
        servo(rmotor,fwd);
        for(x=0;x<255;x++){
        for(y=0;y<255;y++){
        }
        }
}
if(lir>=distf && cir<distf && rir<distf){
        servo(lmotor,fwd);
        servo(rmotor,turn);
```

```
            for(x=0;x<255;x++){
            for(y=0;y<255;y++){
            for(z=0;z<100;z++){
            }
            }
            }
}
if(lir>=distf && cir<distf && rir>=distf){
            servo(lmotor,fwd);
            servo(rmotor,rev);
            for(x=0;x<255;x++){
            for(y=0;y<255;y++){
            }
            }
}
if(lir>=distf && cir>=distf && rir<distf){
            servo(lmotor,fwd);
            servo(rmotor,stop);
            for(x=0;x<255;x++){
            for(y=0;y<255;y++){
            }
            }
}
if(lir>=distf && cir>=distf && rir>=distf){
            servo(lmotor,stop);
            servo(rmotor,stop);
            for(x=0;x<255;x++){
            for(y=0;y<255;y++){
            for(z=0;z<255;z++){
            }
            }
```

```
                        }
                        servo(lmotor,rev);
                        servo(rmotor,rev);
                        for(x=0;x<255;x++){
                        for(y=0;y<255;y++){
                        for(z=0;z<255;z++){
                        }
                        }
                        }
                        servo(lmotor,rev);
                        servo(rmotor,fwd);
                        for(x=0;x<255;x++){
                        for(y=0;y<255;y++){
                        for(z=0;z<500;z++){
                        }
                        }
                        }
                }
        }
}
void trackmode(void){
                        servo(lmotor,fwd);
                        servo(rmotor,rev);
                        delay(9000);

                        servo(lmotor,stop);
                        servo(rmotor,stop);
                        delay(40000);
                }

void trackcolor(){
```

```
while(cir<distc && lir<distc && rir<distc){
uartstring("TC 33 43 23 33 30 90\r");
//outp(cmudat[2],PORTC);
delay(10000);
conv();
if(bump>0x01){bumpswitch();}
        if(cmudat[2]==0x00){
        time--;
                trackmode();                    //tracking color to chase
                if(time<=0x01){
                        time=0x2C;
                        for(e=0;e<400;e++){
                                avoidance();
                        }
                }
        conv();
        }
        else{
if(cmudat[2]>0x00 && cmudat[2]<0x19 && bump<=1){    //track to left
                servo(lmotor,stop);
                servo(rmotor,turn);
                for(x=0;x<255;x++){
                for(y=0;y<255;y++){
                for(z=0;z<300;z++){
                }
                }
                }
        conv();
        }
if(cmudat[2]>=0x19 && cmudat[2]<=0x38 && bump<=1){//track center
                servo(lmotor,fwd);
```

```
                servo(rmotor,fwd);
                for(x=0;x<255;x++){
                for(y=0;y<255;y++){
                }
                }
        conv();
        }
        if(cmudat[2]>0x38 && bump<=1){                  //track to right
                servo(lmotor,turn);
                servo(rmotor,stop);
                for(x=0;x<255;x++){
                for(y=0;y<255;y++){
                for(z=0;z<100;z++){
                }
                }
                }
        conv();
        }
        }
    }
}
void runandhide(){
    while(lir<distf && cir<distf && rir<distf && bump<=0x01 &&
hidemode==1){
        if(cphoto>darkhole && rphoto>darkr && lphoto>darkl){
//no dark spot found
        servo(lmotor,fwd);
        servo(rmotor,fwd);
        for(x=0;x<255;x++){
        for(y=0;y<255;y++){
        }
```

```
                    }
                    conv();
                    }
                    if(cphoto>darkhole && rphoto<=darkr && lphoto>darkl){
                            while(cphoto>darkhole && rphoto<=darkr &&
lphoto>darkl && lir<distf && cir<distf && rir<distf && bump<=0x01){
                            servo(lmotor,fwd);
                            servo(rmotor,turn);
                            for(x=0;x<255;x++){
                            for(y=0;y<255;y++){
                            for(z=0;z<100;z++){
                            }
                            }
                            }
                            conv();
                            }
                    }
                    if(cphoto>darkhole && rphoto>darkr && lphoto<=darkl){
                            while(cphoto>darkhole && rphoto>darkr &&
lphoto<=darkl && lir<distf && cir<distf && rir<distf && bump<=0x01){
                            servo(lmotor,turn);
                            servo(rmotor,fwd);
                            for(x=0;x<255;x++){
                            for(y=0;y<255;y++){
                            for(z=0;z<100;z++){
                            }
                            }
                            }
                            conv();
                            }
                    }
```

```
if(cphoto>darkhole && rphoto<=darkr && lphoto<=darkl){
servo(lmotor,fwd);
servo(rmotor,fwd);
for(x=0;x<255;x++){
for(y=0;y<255;y++){
}
}
while(cphoto>darkhole && rphoto<=darkr &&
lphoto<=darkl && lir<distf && cir<distf && rir<distf && bump<=0x01){
if(rphoto>lphoto){
servo(lmotor,turn);
servo(rmotor,fwd);
for(x=0;x<255;x++){
for(y=0;y<255;y++){
}
}
conv();
}
if(rphoto<lphoto){
servo(lmotor,fwd);
servo(rmotor,turn);
for(x=0;x<255;x++){
for(y=0;y<255;y++){
}
}
conv();
}
if(rphoto==lphoto){
servo(lmotor,fwd);
servo(rmotor,fwd);
for(x=0;x<255;x++){
```

```
                    for(y=0;y<255;y++){
                    }
                    }
                    conv();
                    }
                }
            }
            if(cphoto<=darkhole){
//darkest spot found
            outp(0xFF,PORTC);
            servo(lmotor,fwd);
            servo(rmotor,fwd);
            for(x=0;x<255;x++){
            for(y=0;y<255;y++){
            for(z=0;z<255;z++){
            }
            }
            }
            servo(lmotor,stop);
            servo(rmotor,stop);
            for(x=0;x<255;x++){
            for(y=0;y<255;y++){
            for(z=0;z<255;z++){
            }
            }
            }
            servo(lmotor,fwd);
            servo(rmotor,rev);
            for(x=0;x<255;x++){
            for(y=0;y<255;y++){
            for(z=0;z<500;z++){
```

```
                    }
                    }
                    }
                    servo(lmotor,stop);
                    servo(rmotor,stop);
                    for(x=0;x<255;x++){
                    for(y=0;y<255;y++){
                    for(z=0;z<5000;z++){
                    }
                    }
                    }
                    servo(lmotor,fwd);
                    servo(rmotor,fwd);
                    for(x=0;x<255;x++){
                    for(y=0;y<255;y++){
                    for(z=0;z<500;z++){
                    }
                    }
                    }
                    hidemode=0;
                    conv();
                    }
            }
    }
    int main(void)
    {                       //set up PWM//
    outp(0xFF,DDRD);    //PORTD set output signal for servos left(PD4), right(PD5),
flingage(PD7)
    outp(0xFF,DDRB);    //PORTB set output signal for dispenser(PB3)
    outp(0x00,DDRA);
    outp(0xFF,DDRC);    //set outputs for leds
```

```
        outp(0x64,TCCR0);    //enable PWM in 8bit timer/counter control register0
        outp(0x66,TCCR2);    //enable PWM in 8bit timer/counter control register2
        outp(0xA1,TCCR1A);          //8 bit PWM//
        outp(0x04,TCCR1B); //prescale clock by 256//
        outp(0x80,ADCSR);   //turn on A/D
        sbi(ADMUX,ADLAR);          //left adjust the A/D conversion
                uartinit();
                delay(10000);
                uartstring("PM 1\r");
                delay(1000);
                uartstring("RM 3\r");
                delay(1000);
                sei();
                delay(1000);
                calibrate();
                while(1){
                conv();
                        if(hidemode==0){
                                trackcolor();
                                dispensecandy();
                        }
                        else{
                        if(hidemode==1){
                                runandhide();
                                avoidance();
                        }
                        }
                }
}
```