Ronic
Formal Report
Alexis Mesa
EEL 4665/5666
Intelligent Machines Design Laboratory
TAs : Mike Pridgen, Tim Martin
Instructors: Dr. A. Antonio Arroyo, Dr. Eric M. Schwartz

Table of Contents

Abstract

In any college house party a lot of people face the problem of leaving interesting conversations because they have to re-fill the beverage they are drinking. Once they get back, the seat might have been occupied or the conversation might have died off. While being sitting it would be ideal to put an empty cup on the floor and have it automatically refilled by an autonomous robot with out interrupting the conversation. This boot will be able to avoid standing people and will randomly search for an empty cup which might need a re-fill. However due low funds only the privileged students with the right colored cups will be refilled with higher quality refreshments carried by the robot. Hence the robot will identify a cup and deduce if a qualified person will get a refill then dispense the refreshment and will start looking again for another empty cup. Due to the risk of leaking liquids shortening some devices, the robot will dispense sugar in order to sweeten iced tea.

Executive Summary

Ronic was design to look for specific colored cups and deliver sugar with out knocking out the cup or crashing into any other obstacles. In order to achieve these specifications a PVR board was used with an XMEGA128A1 uP. For actuation hacked servos were used with smoothing functions in order to prevent jerky motions. The obstacle avoidance is one of the most reliable parts of the robot and it uses 3 IR sensors in order to avoid objects in the front and on each side. To minimize wrong turns the obstacle avoidance software uses a triggered IR value and threshold values of IR sensors that are not triggered but are closed to be in order to make a reasonable decision. In case of an IRs become loose during operation or any of them just fail a fall back bump sensors system is used in a parallel configuration in order to save AD channels.

The camera is the most convoluted sensor since it uses RS232 serial connection and packets are sent and receives at a high baud rate. The camera came stripped down and had to be refurbished with a level shifter and a couple of capacitors. All the cup identification and navigation is done with the camera since it has been reliable once the object is indentified. The delivery of the sugar is done with a servo that has a 180 turning radius which functions much like a salt shaker.

Introduction

Today there are many processes that are routinely, wasteful, but necessary which could be solved by the use of automatic systems. Systems such as an autonomous vacuum cleaner or an autonomous pool cleaning robot are advancements which were unheard off a couple of decades ago. However, the field of service is an expanding area where autonomous robots are meant to perform well. Specifically there have been autonomous bartender robots which are able to mix drinks and serve a customer. Others such as “RoboFridge” are simple fridges mounted on wheels which dispense soda to determined clients. In a college town there is an opportunity of automation on parties so that whenever there is a cup on the ground, an autonomous system would re-fill container with a predetermined substance. In order to perform these actives in an efficient manner the robot should be able to avoid obstacles as well as to search for cups that need refills. At the same time only certain types of container should be refilled to avoid distribution of quality refreshment to the wrong client. Hence a mechanism of object avoidance will be necessary and a back up mechanism for extreme cases such as contact is implemented for robustness. A way of recognizing different objects must also be implemented specifically objects with different colors. Finally the robot must be able to successfully dispense the substance and start looking for another cup.

Integrated System

The idea of the robot is to randomly search for a cup with out bumping into any obstacle. Once the container is found, the robot will identify if the cup should be refilled based on color characteristics. If the cup should be refilled the robot would start moving toward the object while avoiding any obstacles and checking if the color “has not changed color”. This last step will avoid the robot to initially recognizing a cup which could be changed while traversing the determined path. Once the robot is with in an acceptable distance from the cup then the substance is delivered. After that, the robot will proceed to randomly look for another cup. The behavioral flowchart is seen on Figure 1.

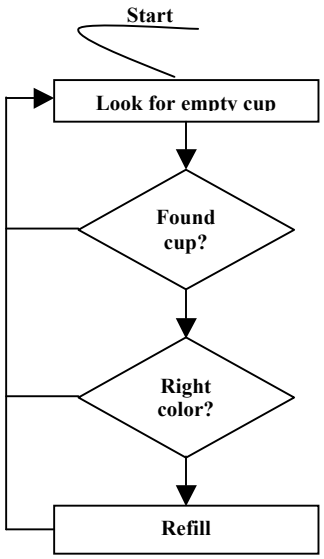

Figure 1

Behavioral Flowchart: The arrows going down are TRUE while the arrows going sideways are FALSE

Mobile Platform

Considering the task, the platform does not have to be extremely fast to accomplish the assignment. However, obstacle avoidance, identification of the cup and successfully refill a cup are with in the specifications requirement.

In order to meet the initial specifications there must be certain hardware to perform the following tasks:

Actuation:

There are 2 hacked servos in order to move on two wheels with a free third wheel for support. This would give the robot the capability to move forward, backwards and rotate 360 degrees on any of its wheel.

Color recognition:

There is a CMUCAM1 on top of the robot in order to recognize a blob determine its color and give the centroid of blob.

Obstacle avoidance:

There are three SHARP IRs which will give signals to the processor if there is something close to the robot on the front part. For completeness there will be bump sensors all around the robot in case the IR fails to recognize a wall or if the wall outside the scope of the IRs (behind the robot) and the robot bumps into it.

Proximity to cup:

There is an IR in the lower middle part of the robot in order to recognize how close the cup is to the robot. Once close enough to the cup the robot will stop and dispense the substance.

Dispenser of sugar:

There is a servo which opens delivers the substance

A more detailed sensor flowchart is seen on Figure 2.

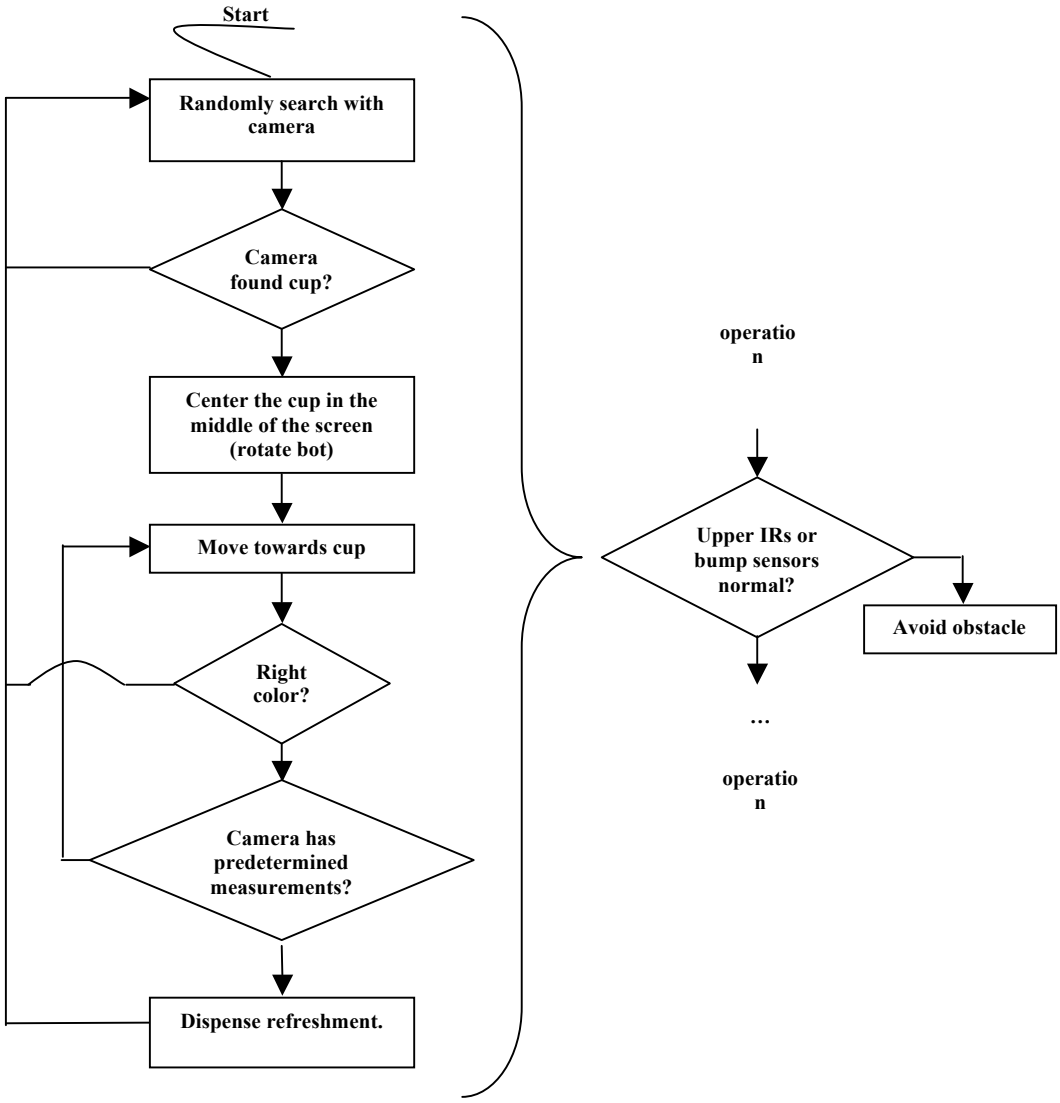

Figure 2

Sensor behaviors: The IRs are used to obstacle avoid which is depicted by the rhombus seen on the side. This runs in after each state on the left half is completed. The left half represents the camera search pattern.

Actuation

In order to move the robot there are 2 wheels driven by continuous rotation servos and a free wheel attached to the back of the chassis for support. This design allows for easy 360 rotation of the robot pivoted in one wheel while lowering design complexity. The 2 motors are hacked HS 322HD servo motors which have the following specifications:

- ✓ Stall Torque: 3.7kg/cm
- ✓ Voltage: 6V
- ✓ Stall Current: 800mA

The reason for the selection of HS 322HD was based on the following assumptions:

- ✓ The robot should weight at most 1 kilogram
- ✓ The wheels are made out of rubber with a radius of 3.5 cm.
- ✓ The robot will only move on horizontal plane made out of asphalt or lower
- ✓ friction coefficient material.
- ✓ The stall torque of each motor is 3.7 kg/cm.

Assuming a coefficient of friction of .8 (rubber to asphalt) then:

Force of friction= F(f)= .8 * 1kg * 9.81m/(s^2)= 7.48 N

Torque(N m)= F(f) * 3.5 cm wheels= .2618 Nm

Torque(kg cm) = (.3531 Nm) * (1m/100cm) * (1kg/9.81N)= 2.67 Kg cm.

Hence one motor could move the whole robot with out stalling. The use of two motors would guarantee that the robot will not stall on a flat surface.

Also there is another motor needed to deliver a substance. A 1.4kg/cm output torque servo was picked due to low cost as well as high stalling torque.

Sensors

There are 3 types of sensors on this robot.

Sharp GP2D120XJ00F IR:

According to datasheet specifications the IR will saturate at 3.1 V with distance from the object of 3 cm and will reflect on objects as far as 40cm with a voltage of .3 V.

There are 2 Sharp GP2D120XJ00F IRs for obstacle avoidance:

These IRs are positioned at the front of the robot in order to locate and avoid obstacles closer than 10 cm.

1 Sharp GP2D120XJ00F IR for recognition of the cup.

With a 3 cm resolution on the lower end of the spectrum the SHARP IR is ideal since a SOLO cup has a top diameter of 9.52 cm. By moving the faucet over the center of the cup there will be a clearance of +/-4.76 cm which is higher than the resolution obtained by the SHARP IR.

Data:

An IR test was conducted to confirm distance vs voltage relation. The test was conducted under fluorescence light conditions and compared with the datasheet values as seen in Figure 3. Even though there are some errors anything from 2 inches to 10 inches are reliable measurements.

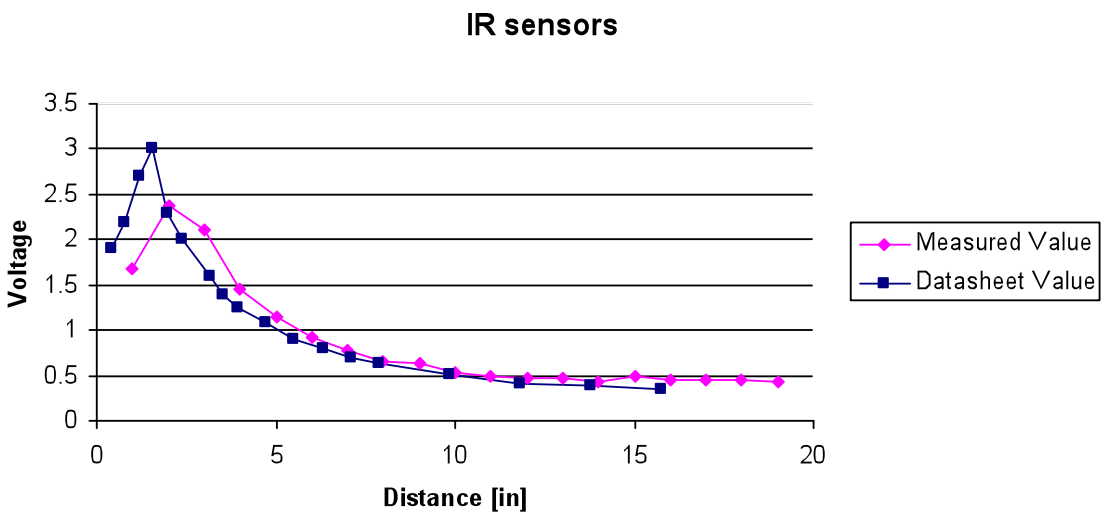

Figure 3

IR sensor test: Comparison between the values obtained from the uP to the values provided by the datasheet

Bump switches

For completion there are 5 bump switches distributed equally on the front of the robot in case the IRs fails to recognize a wall or in case the robot is moving in a scope outside the range of vision of the IRs.

Data:

In order to make a smart bump switch with minimum valuable resources such as ADC, all the switches are connected as in Figure 4. This configuration will give different voltages to each switch if pressed making them unique.

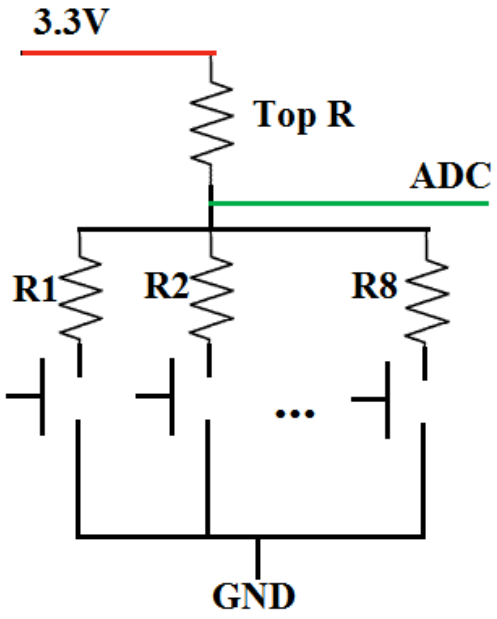

Figure 4

Parallel configuration of resistors with different resistance value makes each switch have a unique voltage value when turn on

Further testing was made by measuring the value with a voltmeter and comparing the AD value of the uP. As seen on Figure 5, there are some errors however they are tolerable.

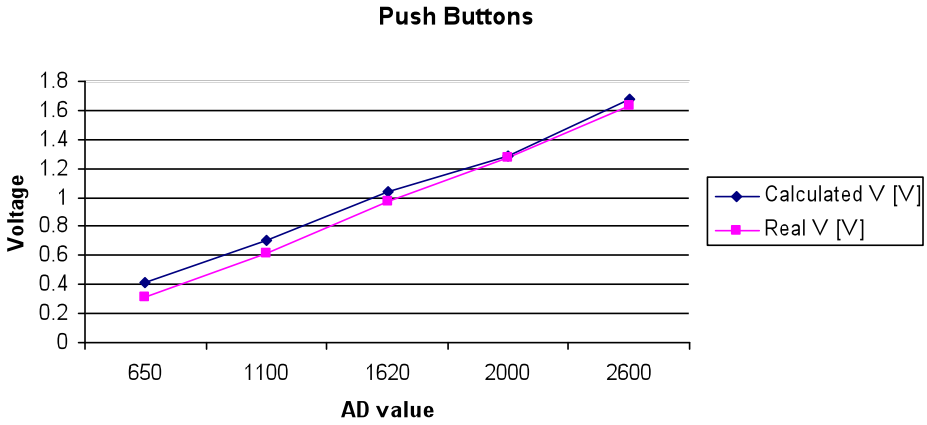

Figure 5

Comparison between the voltage level measured with a voltmeter and the AD value from the uP

Camera

A CMUCAM1 from the BoeBot is used to perform the following tasks:

- ✓ Identify a blob
- ✓ Identify the color of the blob
- ✓ Indicate direction of the blob at long and medium range distance.

This camera comes with features that are able to fulfill these requirements such as:

- ✓ Find the centroid of any tracking data
- ✓ Gather mean color and variance data
- ✓ Track user defined color blobs at up to 50 Frames Per Second.

Top level Overview:

The CMUCAM has the following operation block diagram:

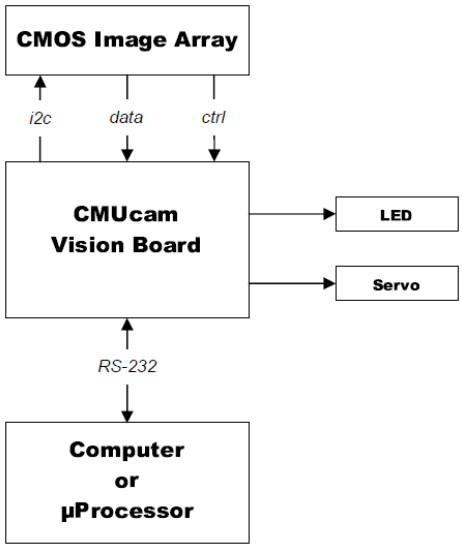

Figure 6

Top level overview, image provided from CMUCAM manual.

Electrical requirements:

The board works with an input of 9V~6V with an LDO regulator of 5V.

There are 2 ways of communicating to the board:

- ✓ 5V Transistor-Transistor Logic (TTL)
- ✓ 3.3 V RS232 which is achieved by a level shifter (MAX232)

Servo port does not go trough the regulator of the board which is a major pitfall however in this project this port is not used.

Board assembly:

The BoeBot uses TTL in order to communicate to its processor hence there is no level shifter (MAX232) on the board. However, the PVR board uses RS232 and in order to focus the camera with the java applet provided by The Robotics Institute at Carnegie Mellon University. For this reason the blue additions on Figure 7 were made in order to use the RS232 level shifted serial port seen in green.

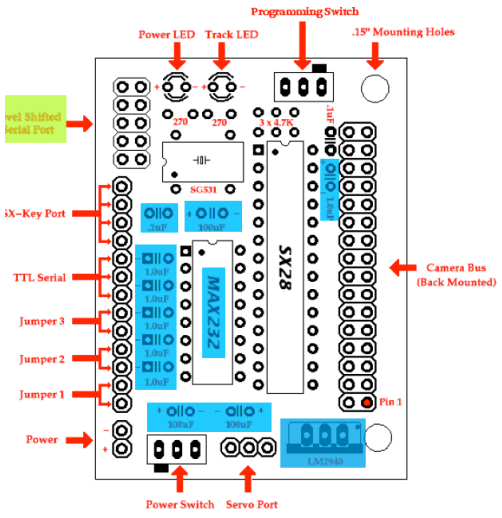

Figure 7

Additions made to the board are highlight in blue. These are: MAX232 level shifter, six 1 uF capacitors, three 100uF capacitors and a .1uF capacitor.

Communication to the board:

There are two ways of communicating to the board:

TTL: This avoid the MAX232 level shifter

RS232: Level shifted serial output.

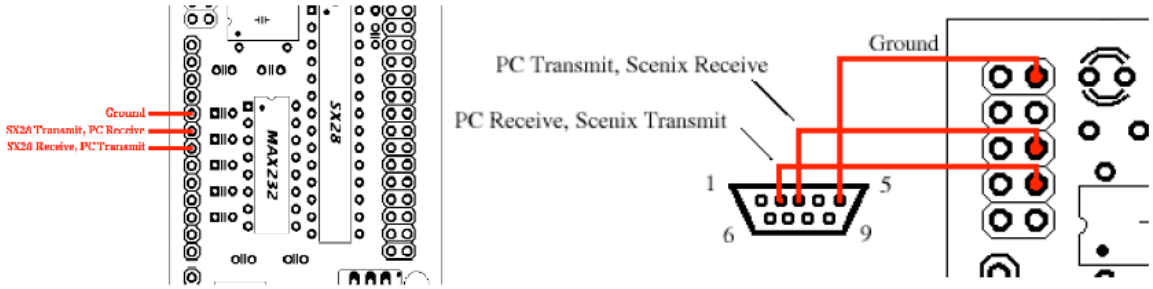

Figure 8

Left figure is TTL connection while right figure is an RS232 connection

The option explored in this project is the RS232 serial connection.

The serial specifications required by the CMUCAM are:

- ✓ 115,200 Baud
- ✓ 8 Data bits
- ✓ 1 Stop bit
- ✓ No Parity
- ✓ No Flow Control (Not Xon/Xoff or Hardware)

Baud rate is selected as follow by toggling Jumper 2 and 3

Baud Rate	Jumper	Position	Jumper	Position
115200	2	Open	3	Open
38400	2	Set	3	Open
19200	2	Open	3	Set
9600	2	Set	3	Set

Figure 9

Baud rate selection

Jumper 1 is used to run the camera as Slave mode and Jumper 4 is used to run a Demo program.

All commands sent and received from the camera are ASCII characters (so that 123 are 3 bytes “1” ,“2”, “3”). When a successful transmition of a command occurs an ACK string is returned else a NCK string is returned. After both of these return string an “\r” is returned followed by a ”:” which means that the camera is in the idle state and is waiting for a command. Spaces do not matter since they separate arguments. The ASCII “\r” is sent to the camera at the end of each command. There is a wide repertoire of commands which range from get mean color value to get current version of firm ware.

Testing the Camera:

In order to test the camera these steps were followed.

- ✓ Assemble board as recommended in Figure 5.
- ✓ Power up with a 9V~6V supply, red LED should turn on.
- ✓ Power off.
- ✓ Construct a serial cable to connect to serial port of computer.
- ✓ Open HyperTerminal with 115,200 Baud, 8 Data bits, and 1 Stop bit.
- ✓ Connect as seen in Figure 8 (right) and power on board. The following should come up on the terminal:

  >>CMUcam v1.12

Focusing the camera:

Use the java program provided by The Robotics Institute at Carnegie Mellon University. As seen in Figure 10, focusing the camera is an important procedure which completely alters its functionality.

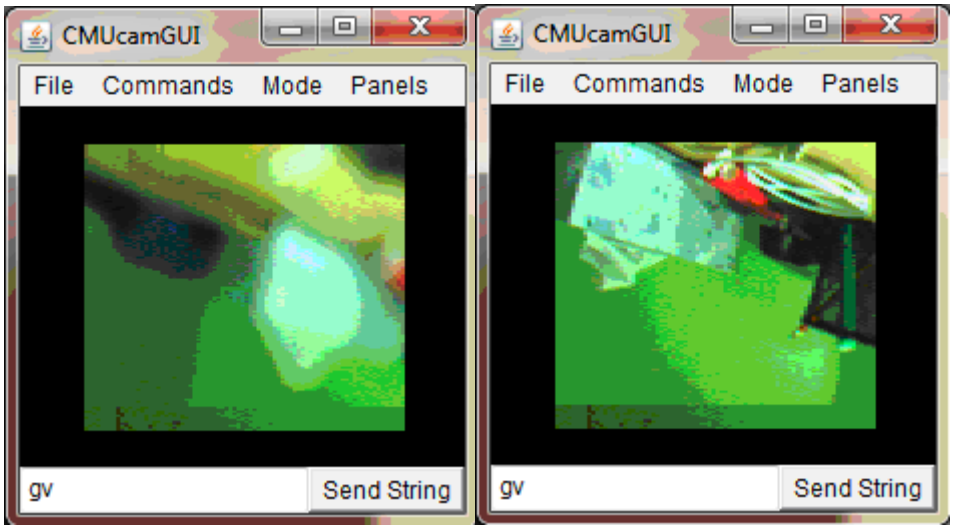

Figure 10

Left unfocused camera while focused camera is showed on the right. Focusing is made by unscrewing or screwing in the lens.

Use of the camera:

The camera is used to locate red colored LEDs. This was achieve by:

- ✓ Resetting the camera
- ✓ Enable white balance and auto gain for 5 seconds in order to adjust for current lighting conditions. Once adjusted, they are turned off in order to obtain a faster response from the camera.
- ✓ Enable the tracking LED so that there is a visual representation if the camera sees something with in the parameters given (red color).

- ✓ Enable polling mode. This returns exactly one package if a image processing function is called.
- ✓ Enable Middle of Mass which returns the middle of mass x value, the middle of mass y value, the left most corner's x value, the left most corner's y value, the right most corner's x value, the right most corner's y value, number of Pixels in the tracked region, scaled and capped at 255: (pixels+4)/8, and the confidence which is the (# of pixels / area)*256 of the bounded rectangle and capped at 255. However only the x and y coordinates are used for location.

Behaviors

The robot uses the following behaviors: obstacle avoidance, search of the cup and smaller subsequent behaviors.

Obstacle avoidance

This behavior uses 3 IR sensors: one in the front (middle) to avoid obstacles straight ahead, and 2 oriented 45 degrees from the center of the robot. The software is quite simple since it polls each IR sensor continuously and reacts upon each reading. If the middle IR detects that there is something on the way then the robot will move in the direction of the side IR with the lowest AD value in order to prevent the robot turning into another side obstacle as seen in Figure 10. While turning it waits until one of the side IRs gets trigger to move forward again or until a timer reaches certain value in case the side IR never reads anything. If the side IR gets triggered then the robot rotates toward its middle continuing the motion from the middle IR sequence.

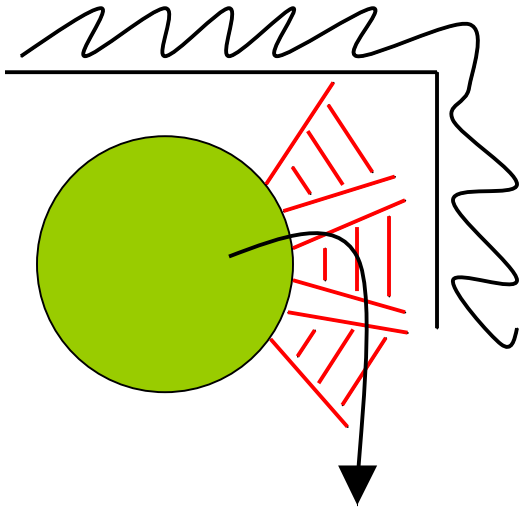

Figure 10

If the front IR gets triggered then the 2 side IRs values are compared the robot steers towards the IR with the lower value. Once the middle IR is not triggered anymore the IR towards the wall gets triggered to continue the motion.

Cup searching pattern

The robot wanders around until it finds a cup with a predetermined color. Once the camera determines that there is a color on its field of vision the robot is rotated so that the center of mass of the cup lies in the middle of the horizontal image. Once this task is accomplished, the robot starts moving towards the cup while obstacle avoiding and adjusting the direction of the robot if the cup changes place or if the robot deviates from a straight line. The adjustment is done with a low level of precision so that the robot does not have to correct so often. As the robot gets closer to the cup, the robot slows down and the angle of vision gets narrower so that precision increases. This is seen on Figure 11. Once the robot is closed enough to activate the front IR sensor the obstacle avoidance software is turned off and the robot keeps moving forward until a predetermined vertical value (of the camera center of mass)is reached and the sugar is delivered.

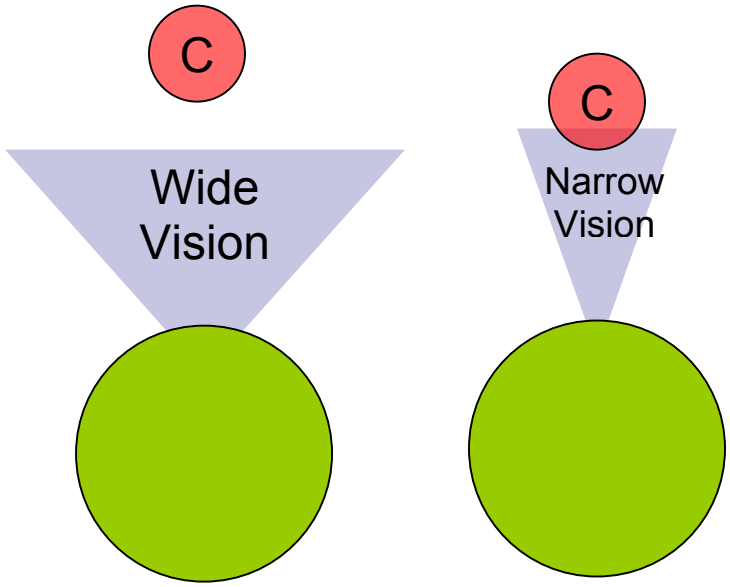

Figure 11

When the cup is far away the robot does not correct as much as when the cup is closed by to avoid jerky movements.

Slowing the robot

In order to get consistent results (not jerky motions which blurs out the camera image) the hacked servos had to be slowed down in a smooth matter. In order to do this the following function was used:

Intermediate Value= 10.0/(10.0+1.0)*old_value + 1.0/(10.0+1.0)*new_value;

Results are seen on Figure 12 which depicts a sweep from 300 to 1200 (native values for the servo used on the project).

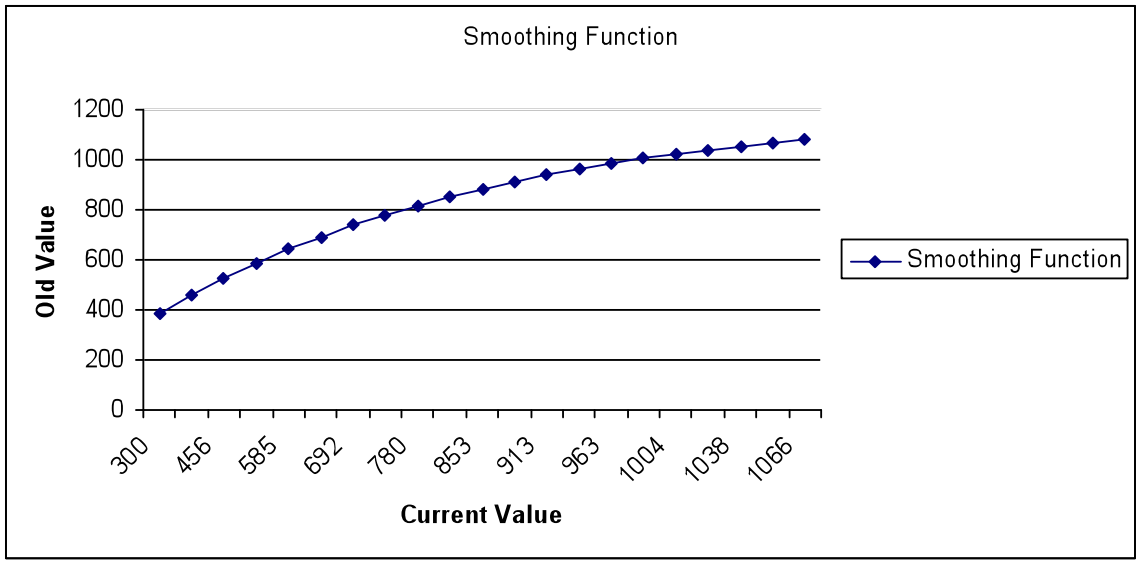

Figure 12

A smoothing function which other wise would look like a step function, the smoothing aspect of this functions prevents the robot to have jerky turns.

Sugar delivery

The initial idea had a funnel with a tube going to the front of the robot which dispensed the sugar. However, since sugar is of thick granule the tube was not letting the grains flow in a continuous matter. Since a simpler solution always beats any overly complicated solution a servo was attached to the front of the robot which simply rotates a small box full of sugar back and forward (much like a salt shaker).

Experimental Layout and Results

After calibration, the robot ran 3 different tests:

Obstacle avoidance:

By just activating the obstacle avoidance software the robot was able to avoid all of the boxes seen on Figure 13. The trial lasted for 3 minutes and the results are the following:

It knocked the box with the red arrow once since it did not turn quiet fast and all of the boxes are empty and easy to knock down.

It passed in between the boxes pointed with black arrow with a clearance of 2 inches with out touching them 3 times.

Figure 13

Ronic was able to passé between the 2 black arrowed boxes with a clearance of only 2 inches! Unfortunately it knocked over the smaller box pointed by the red arrow.

Cup recognition:

A 9 trial test was run with the CMUCAM recognizing the cup and the obstacle avoidance software turned off. The robot was successful in all the trials to deliver the substance with in the cup dimension as seen in Figure 14

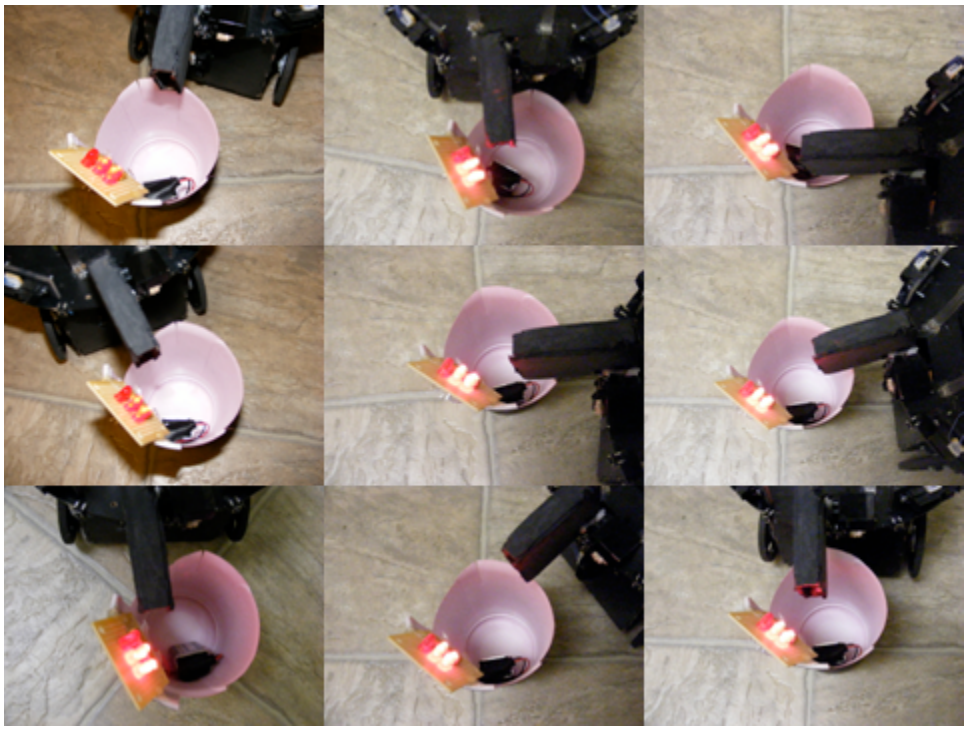

Figure 14

These are 9 different trials of delivering the sugar to the cup. Even though not all of the deliveries were right in the middle of the cup, all of them were successfully within in the cup boundaries.

Cup recognition+ obstacle avoidance:

Another test for 3 minutes was run with the same set up as Figure 14 but with both behaviors integrated. The results were similar than before, the robot recognized the cup 4 times and did not knock out any of the boxes.

Lesson learned

"Making the simple complicated is commonplace; making the complicated simple, awesomely simple, that's creativity"

-Charles Mingus

1. That was the biggest lesson learned. The best solutions are the one that seem relatively easy to accomplish such as the obstacle avoidance.
2. Also another lesson learned is to CAD the whole design instead of most of the parts. The few components that were not done in Solidworks were the only ones I had issue with.
3. Make back ups and different revisions of code and presentations.
4. Hacked servos are limited in many ways. Power is limited, speed is limited, and even controlling them accurately is quiet hard to do.
5. Important connections such as JTAG, power switch, and other switches must be access relatively easily.
6. AD converters of uP are quiet inaccurate.

Conclusion

The robot was able to fulfill all the specifications made at the beginning of design. Obstacle avoidance is one of the best features since it rarely if ever fails and is able to change the path of the robot to enter spaces with just 2 inches of clearance (which was not a requirement but a byproduct of the software). The camera although limited (due to lighting conditions) is a reliable sensor when used correctly. All the orientation of the robot towards the cup is made with the camera and has never failed to track an object once identified. Further improvement is necessary on the physical aspect of the robot. A new platform that will solve some of the problems of the previous section will improve increase functionality. On the other hand the electronic aspects of the project seem to be reliable and robust with the only addition being replacing the servos.

Documentation

Pridgen Vermeer Robotics Xmega128 Manual

AVR1000: Getting Started Writing C-code for XMEGA

http://www.atmel.com/dyn/resources/prod_documents/doc8075.pdf

AVR XMEGA A1 Device Datasheet

http://www.atmel.com/dyn/resources/prod_documents/doc8067.pdf

AVR1306: Using the XMEGA Timer/Counter

http://www.atmel.com/dyn/resources/prod_documents/doc8045.pdf

XMEGA A MANUAL

http://www.atmel.com/dyn/resources/prod_documents/doc8077.pdf

AVR1300: Using the XMEGA ADC

http://www.atmel.com/dyn/resources/prod_documents/doc8032.pdf

CMUCAM1 user manual

http://www.cs.cmu.edu/~cmucam/Downloads/CMUcamManual_1_8.pdf

Specification of hs-322hd Standard Deluxe Servo

http://www.robodacta.com.mx/UserFiles/File/HS322HD.pdf

Sharp specifications

http://www.sparkfun.com/datasheets/Sensors/Infrared/GP2D120XJ00F_SS.pdf

Appendices

Appendix A: Photos

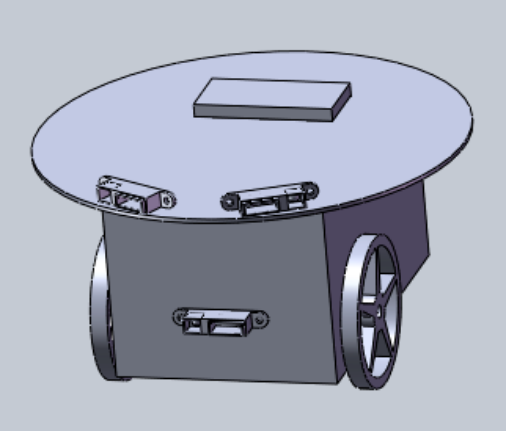

Figure 15

Initial CAD design done in Solidworks.

Figure 16

Initial results model with hacked servos and LCD mounted.

Figure 17

All components mounted except the sugar delivery servo.

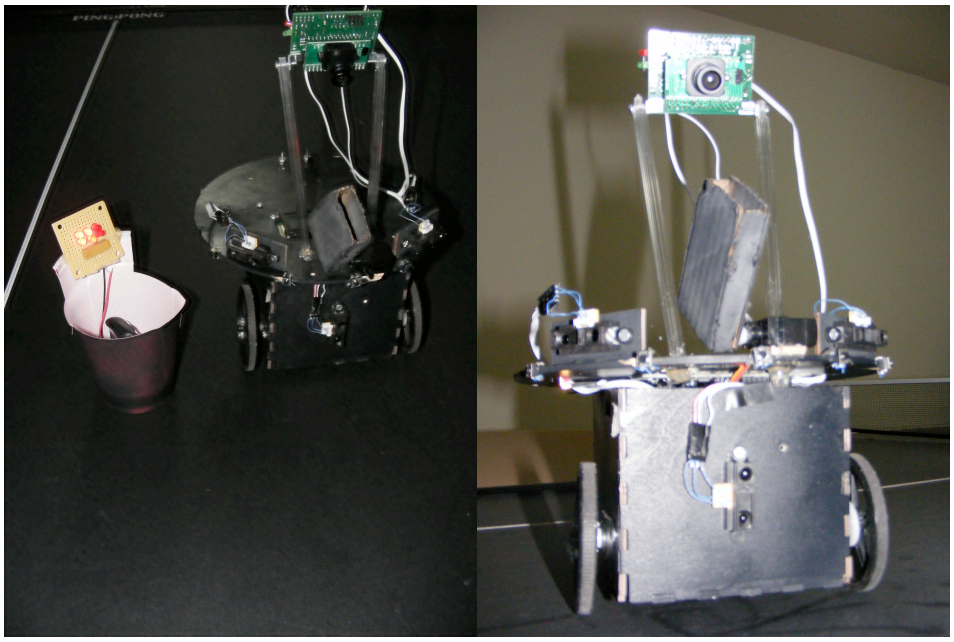

Figure 17

Final model; sugar delivery servo mounted in the middle and CMUCAM is raised from previous design

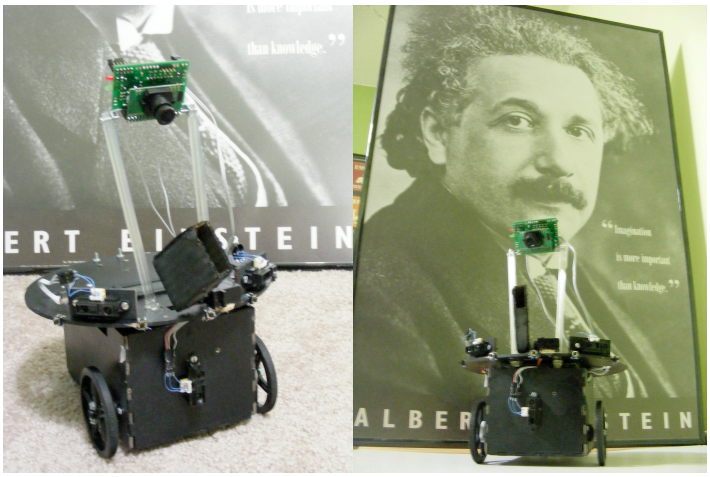

Figure 18

Other views

Figure 19

Identifying the cup, delivering and then moving away.

Appendix B: Code

Main code:

```
**********************************************************************************
#include <avr/io.h>
#include "PVR.h"
#include <stdbool.h>
int forward=300;
int forward_slow=725;
int backward_slow=775;
int backward=1200;
int stop=750;
int down=400;
int up=700;

void main(void)
{
        int IRleft, IRright, IRmiddle;
/***************
Initializations
***************/
        xmegaInit();                                                    //setup XMega
        delayInit();                                                    //setup delay functions
        ServoCInit();                                                   //setup PORTC Servos
        lcd_init();                                                     //Setup LCD
        ADCAInit();                                                     //setup PORTA
analong readings
        PORTQ_DIR |= 0x01;                                              //set Q0 (LED) as output
        delay_ms (1000);
        PORTB_DIR = 0x30;                                               //This is setting the pin 0 of port B to output
a 3.3 V so that this could be used as VREF for the ADC
        lcd_clear();
        PORTE_DIR   = PIN3_bm;                  //Pin 3 of port E is output
        PORTE_OUT   = PIN3_bm;                  //Pin 3 of port E is TXO
        PORTE.DIRCLR   = PIN2_bm;       //Pin 2 of port E is RXO
        USARTE0_CTRLC = 0x03;                   // USART Control Register C: ASYNCHRONOUS, no parity 1
stop bit 8 bit word
        USARTE0_BAUDCTRLA = 0x06;               // Page 238 of Atmel manual, fbaud=115200 (my
case)=32MHz/(16*(((2^BSCALE) * BSEL)+1))
        USARTE0_BAUDCTRLB = 0xC1;               // BSEL= 262 and BSCALE= -4 in 2s comp .
        USARTE0_CTRLB |= 0x08;                  // TX0 is on
        USARTE0_CTRLB |= 0x10;                  // RX0 is on
        static char *temp;                      //String pointer where the returned packets are going to be stored


/****************************
Program, set up the camera
to track red colored objects
****************************/

CMUsend("RS\r");    //Reset the camera
delay_ms (500);     //Long enough to wait for ACK and to get the camera ready
CMUsend("CR 18 44 19 33\r");  // Run white balance and auto gain for 5 sec
delay_ms (5);
CMUsend("CR 18 40 19 32\r");  // Turn them off
temp=CMUreceive(); //Receive ACK
delay_ms(5);
CMUsend("L1 2\r");  //To turn on green light
temp=CMUreceive(); //Receive ACK
delay_ms(5);        //For some reason that I havent figure out, there must be a delay between commands
CMUsend("PM 1\r"); //Activate polling mode
temp=CMUreceive(); //Receive ACK
delay_ms(5);
CMUsend("MM 1\r");          //Activate Middle of mass mode
temp=CMUreceive(); //Receive ACK
delay_ms(5);
int         i=0;
int j=0;
```

```
int itemp=0;
int x=0, y=0;
int retptr=2;                   //Return pointer from used in decoding the packet returned from the camera

/********
Behavior
*********/
        while(1)
        {

                ServoC3(up);            //Sugar is not dispence(up position)

                static char *temp;
                CMUsend("TC 155 255 1 30 1 30\r");      //Track color
                temp=CMUreceive();                      // Receive ACK
                temp=CMUreceive();                      // Receive M packet
                delay_ms(5);
                CMUTC(temp, &x, &y);                    //Decode the M packet
                free(temp);                                     //Free the temp variable


                /***********************
                Display X,Y coordinates.
                ************************/
                lcd_clear();
                lcd_write("x=");
                lcd_write_int(x);       //X value is left and right
                lcd_write(" y=");
                lcd_write_int(y);       //Y value tell you depth


                if(y<80)                //y<80 is the about 10 inches for from the cup or more
                {
                        //Obstacle avoidance code
                        Servo_L_R(forward, forward);
                        IRleft=ADCA4();
                        IRright=ADCA1();
                        IRmiddle=ADCA6();
                        while (IRmiddle>1700)
                        {

                                IRleft=ADCA4();
                                int i=0;
                                while(IRleft<1500 && i<40)
                                {
                                        Servo_L_R(forward, backward);;
                                        IRleft=ADCA4();
                                        i++; //In case it never exits the loop
                                }
                                IRmiddle=ADCA6();
                        }


                        IRright=ADCA1();
                        while (IRright>2000)
                        {
                                        lcd_write("RIGHT");
                                        Servo_L_R(backward, forward);
                                        IRright=ADCA1();
                        }

                        IRleft=ADCA4();
                        while (IRleft>2000)
                        {
                                        lcd_write("LEFT");
                                        Servo_L_R(forward, backward);
                                        IRleft=ADCA4();
                        }

                        //To center the camera with a wide vision.
```

```
                        while(x<28 && x>0)
                        {
                                Servo_L_R(forward_slow, backward_slow);
                                static char *temp;
                                CMUsend("TC 155 255 1 30 1 30\r");      //Track color
                                temp=CMUreceive();                      // Receive ACK
                                temp=CMUreceive();                      // Receive M packet
                                delay_ms(5);
                                CMUTC(temp, &x, &y);
                                free(temp);
                        }
                        //To center the camera with a wide vision.
                        while (x>70 && x<88)
                        {
                                Servo_L_R(backward_slow, forward_slow);
                                static char *temp;
                                CMUsend("TC 155 255 1 30 1 30\r");      //Track color
                                temp=CMUreceive();                      // Receive ACK
                                temp=CMUreceive();                      // Receive M packet
                                delay_ms(5);
                                CMUTC(temp, &x, &y);
                                free(temp);
                        }



                }


                //This is 10 inches or less. The robot slows down.
                else if(y>=80 && y<132)
                {
                        //Narrow vision centered in around x=55
                        if(x<47 && x>0)
                        {
                                Servo_L_R(forward_slow, backward_slow);
                        }
                        //Narrow vision centered in around x=55
                        else if (x>63 && x<88)
                        {
                                Servo_L_R(backward_slow, forward_slow);
                        }
                        else
                        {
                                Servo_L_R(forward_slow, forward_slow);
                        }
                }
                //Dispense sugar!
                else
                {
                        Servo_L_R(forward_slow, forward_slow);
                        delay_ms(100);
                        Servo_L_R(stop, stop);
                        ServoC3(down);
                        delay_ms(4000);
                        ServoC3(up);
                        //Move away
                        Servo_L_R(backward, backward);
                        delay_ms(1000);
                        Servo_L_R(forward, backward);
                        delay_ms(1000);
                        Servo_L_R(forward, forward);
                }
        }
}
```

****

Library/Source File

```
**************************************************************************************
#include <avr/io.h>
#include <avr/interrupt.h>
#include "PVR.h"

/*********
 * Xmega *
 *********/

void xmegaInit(void)
{
	CCP = 0xD8;
	CLK_PSCTRL = 0x00;
	PORTQ_DIR = 0x01;
	//setup oscilllator
	OSC_CTRL = 0x02;                                        //enable 32MHz internal clock
	while ((OSC_STATUS & 0x02) == 0);          //wait for oscillator to be ready
	CCP = 0xD8;                                                                    //write signature to
CCP
	CLK_CTRL = 0x01;                                        //select internal 32MHz RC oscillator
}

/*********
 * Delay *
 *********/

void delayInit(void)
{
	TCF1_CTRLA = 0x01;                                              //set clock/1
	TCF1_CTRLB = 0x31;                                              //enable COMA and COMB, set
to FRQ
	TCF1_INTCTRLB = 0x00;                                   //turn off interrupts for COMA and COMB
	SREG |= CPU_I_bm;                                       //enable all interrupts
	PMIC_CTRL |= 0x01;                                              //enable all low priority interrupts
}

void delay_ms(int cnt)
{
	delaycnt = 0;                                                   //set count value
	TCF1_CCA = 32000;                                       //set COMA to be 1ms delay
	TCF1_CNT = 0;                                                   //reset counter
	TCF1_INTCTRLB = 0x01;                                   //enable low priority interrupt for delay
	while (cnt != delaycnt);                        //delay
	TCF1_INTCTRLB = 0x00;                                   //disable interrupts
}

void delay_us(int cnt)
{
	delaycnt = 0;                                                   //set counter
	TCF1_CCA = 32;                                                  //set COMA to be 1us delay
	TCF1_CNT = 0;                                                   //reset counter
	TCF1_INTCTRLB = 0x01;                                   //enable low priority interrupt for delay
	while (cnt != delaycnt);                        //delay
	TCF1_INTCTRLB = 0x00;                                   //disable interrupts
}

SIGNAL(TCF1_CCB_vect)
{
	delaycnt++;
}

SIGNAL(TCF1_CCA_vect)
{
	delaycnt++;
}

/*******
 * LCD *
 *******/
```

```
/*
                LCD PINOUT

                                        01 - Ground (Vss)
                                        02 - 5V Power (Vcc)
                                        03 - Contrast (Vo)
                                        04 - Register Select (RS)
                                        05 - Read/Write_L (R/W_L). Always writing hence is grounded
                                        06 - Enable (E)
                                        07 - Data 4 (DB4) LSB
                                        08 - Data 5 (DB5)
                                        09 - Data 6 (DB6)
                                        10 - Data 7 (DB7) MSB

                Important for understanding
                                        PIN 4 RS Input 0 = command input/output, 1 = data input/output
                                        PIN 5 R/W Input 0 = write to LCD module, 1 = read from LCD module
                                        PIN6 EN Input enable signal (data strobe)

                Port pin out


                                        PORTD.5 - Enable (E)
                                        PORTD.4 - Register Select (RS)
                                        PORTD.3 - Data 7 (DB7)
                                        PORTD.2 - Data 6 (DB6)
                                        PORTD.1 - Data 5 (DB5)
                                        PORTD.0 - Data 4 (DB4)
*/


/*
Writes commands to the LCD, it stores the 8 bit data into a temp variable, sends upper nibble thne sends lower nibble
*/

void lcd_command(unsigned char data)
{
        unsigned char temp= data;
        PORTD_OUT= 0x00;                                //Clear out input
        delay_ms(5);
        data= ((data & 0xF0)>>4)|0x20;  //Make data = to upper nibble and put enable high and Rs low
        PORTD_OUT= data;
        delay_ms(5);
        data= data & 0x0F;                              //Make E low
        PORTD_OUT= data;
        delay_ms(5);

        PORTD_OUT= 0x00;                                //Clear out input
        data=temp;
        delay_ms(5);
        data= (data & 0x0F)|0x20;               //Make data = to lower nibble and put enable high and Rs low
        PORTD_OUT= data;
        delay_ms(5);
        data= data & 0x0F;                              //Make E low
        PORTD_OUT= data;
        delay_ms(5);
}


/*
Sends data (actual numbers and letters) to LCD the only difference from lcd_command is Rs which now is high
*/

void lcd_data(unsigned char data)
{
        unsigned char temp= data;
        PORTD_OUT= 0x00;                                //Clear out input
        delay_us(50);
        data= ((data & 0xF0)>>4)|0x30;  //Make data = to upper nibble and put enable high and Rs high
        PORTD_OUT= data;
```

```
		delay_us(50);
		data= data & 0x1F;                                  //Make E low
		PORTD_OUT= data;
		delay_us(50);

//Lower nibble

		PORTD_OUT= 0x00;                                    //Clear out input
		data=temp;
		delay_us(50);
		data= (data & 0x0F)|0x30;                   //Make data = to lower nibble and put enable high and Rs high
		PORTD_OUT= data;
		delay_us(50);
		data= data & 0x1F;                                  //Make E low
		PORTD_OUT= data;
		delay_us(50);
}


/*
Initiates LCD: 4 bit mode, 2 lines, display on, cursor on and blinking. At the end it clears and goes back to home
*/

void lcd_init(void)
{
	PORTD_DIR = 0x3F;
	lcd_command(0x33);   //4 bit mode
	lcd_command(0x32);                  //4 bit mode
	lcd_command(0x2C);                  //Enable 2 line mode
	lcd_command(0x0F);   //Display, cursor, blink
	lcd_command(0x01);   //clear home
}


/*
Writes Characters to screen until it gets to the end of the string
*/

void lcd_write(char *str)
{

//		lcd_command(0X80);      // Start Cursor From First Line
		//lcd_com(0XC0);        // Start Cursor From Second Line

		unsigned int i=0;
		for(i=0 ;str[i]!= 0x00 ;i++) //0x00= NULL in ASCII
			lcd_data(str[i]);
}

void lcd_write_int(int value)
{
		int temp;
		temp=value;
		value= value/1000;                          //get the 1000th place
		lcd_data(value+48);   //48 is ascii for 0
		temp=temp-value*1000;                       //save what is left of the value to the temp register.

		value=temp/100;                                     //get the 100th place
		lcd_data(value+48);
		temp=temp-value*100;

		value=temp/10;
		lcd_data(value+48);
		temp=temp-value*10;

		lcd_data(temp+48);
}

void lcd_clear(void)
{
```

```
	lcd_command(0x01);		//clear home
}

/*********
 * Servo *
 *********/

void ServoCInit(void)
{
	TCC0_CTRLA = 0x05;					//set TCC0_CLK to CLK/64
	TCC0_CTRLB = 0xF3;					//Enable OC A, B, C, and D.  Set to Single Slope PWM
								//OCnX = 1 from Bottom to CCx
and 0 from CCx to Top
	TCC0_PER = 10000;				//20ms / (1/(32MHz/64)) = 10000.  PER = Top
	TCC1_CTRLA = 0x05;					//set TCC1_CLK to CLK/64
	TCC1_CTRLB = 0x33;					//Enable OC A and B.  Set to Single Slope PWM
								//OCnX = 1 from Bottom to CCx
and 0 from CCx to Top
	TCC1_PER = 10000;				//20ms / (1/(32MHz/64)) = 10000.  PER = Top
	PORTC_DIR = 0x3F;				//set PORTC5:0 to output
	TCC0_CCA = 0;					//PWMC0 off
	TCC0_CCB = 0;					//PWMC1 off
	TCC0_CCC = 0;					//PWMC2 off
	TCC0_CCD = 0;					//PWMC3 off
	TCC1_CCA = 0;					//PWMC4 off
	TCC1_CCB = 0;					//PWMC5 off
}

/*
Takes left value and right value (see servo 2 for more instruction in timing)
Since left servo is backward then the values must be reversed.
Then a smoothing function is applied. With a 20ms delay whihc will take a total of 50*20ms=1 second to complete

Since the TCC0_PER (period) is set to 20 ms or a count of 10000 thne
10000 ------- 20 ms
  x   -------  x ms
*/

void Servo_L_R(int Lvalue, int Rvalue)
{
	int i, old_Rvalue, new_Lvalue, new_Rvalue;
	float old_Lvalue;
	float new_L_value;

	new_Lvalue=Lvalue+10;		//The left servo does not stop at the right value hence an offset must be used


	if (Rvalue==300)		//Reverse values
	{
		new_Rvalue=1200;
	}
	else if (Rvalue==1200)		//Reverser values
	{
		new_Rvalue=300;
	}
	else if (Rvalue==725) //Reverser values
	{
		new_Rvalue=775;
	}
	else if (Rvalue==775) //Reverser values
	{
		new_Rvalue=725;
	}
	else
	{
		new_Rvalue=Rvalue+5;
	}
	//Smoothing function
	for (i=0; i<50 ; i++)
	{
```

```
                        old_Lvalue= TCC0_CCB;
                        old_Rvalue=TCC0_CCC;
                        TCC0_CCC= 10.0/(10.0+1.0)*old_Rvalue + 1.0/(10.0+1.0)*new_Rvalue;
                        TCC0_CCB=(10.0/(10.0+1.0))*old_Lvalue + (1.0/(10.0+1.0))*new_Lvalue;
                }

}

void ServoC2(int value)
{

                TCC0_CCA = (value);                     //Generate PWM.

}
void ServoC3(int value)
{

                TCC0_CCD = (value);                     //Generate PWM.

}
void ServoC4(int value)
{

                TCC1_CCA = (value);                     //Generate PWM.

}
void ServoC5(int value)
{

                TCC1_CCB = (value);                     //Generate PWM.

}


/**********
 * CMUCAM *
 **********/

void CMUsend(char *command)
{
                int i = 0;
                while (command[i] != '\0')              //While command does not end do...
                {

                        while (!(USARTE0_STATUS & (1<<USART_DREIF_bp)));        // Data Register Empty Flag: check if data
register is empty
                        USARTE0_DATA = command[i];
                //USARTE0_DATA is shared by the transmit and receive
                        while (!(USARTE0_STATUS & (1<<USART_TXCIF_bp)));        // if data is sent TXCIF is set
                        i++;
                }

}

char* CMUreceive(void)
{

                int i=0;
//              static char data[20];
                static char* data;
                static char* data2;
                data=malloc(20*sizeof(char));
                data2=data;

                do
                        {

                                while (!(USARTE0_STATUS & (1<<USART_RXCIF_bp))){}       // wait for receive is complete
                                *(data2++) = USARTE0_DATA;
                //Put data into string
```

```
                        } while (data[i++] != '\r');                                    //Since all
trasnfers are ended by \r wait for it to happen

        return data;

}
/*
The M Packet the camera returns is formatted as follow
middle of mass x value
space
the middle of mass y value
space
the left most corner's x value
space...
the left most corner's y value
...
the right most corner's x value
...
the right most corner's y value
number of Pixels in the tracked region scaled and capped at 255:  (pixels+4)/8,
the confidence which is the (# of pixels / area)*256 of the bounded rectangle and capped at 255
However only the x and y coordinates are used for location.

*/
void CMUTC(char *temp, int *x, int *y)
{

        int retptr=2;

        (*x)=temp[retptr]-48; //-48 is ASCII for 0
        retptr++;
        if (temp[retptr]!=' ')    //if it is not a space is a 2 digit number
        {
                (*x)=(*x)*10+temp[retptr]-48;
                retptr++;
        }
        retptr++;
        (*y)=temp[retptr]-48;
        retptr++;
        if (temp[retptr]!=' ')    //If it is not a space is a 2 digit or 3 digit number
        {
                (*y)=(*y)*10+temp[retptr]-48;
                retptr++;
                if (temp[retptr]!=' ')    //If it is not a space is a 3 digit number
                {
                        (*y)=(*y)*10+temp[retptr]-48;
                        retptr++;
                }
        }
}

/********
 * ADCA *
 ********/

void ADCAInit(void)
{
        delayInit();
        ADCA_CTRLB = 0x00;                                      //12bit, right adjusted
        ADCA_REFCTRL = 0x30;                            //set to Vref = 1.0V (approx)
        ADCA_CH0_CTRL = 0x01;                           //set to single-ended
        ADCA_CH0_INTCTRL = 0x00;                //set flag at conversion complete.  Disable interrupt
        ADCA_CH0_MUXCTRL = 0x08;                        //set to Channel 1
        ADCA_PRESCALER = 0x03;                          // slow down conversion
        ADCA_CTRLA |= 0x01;                                     //Enable ADCA
        PORTB_DIR = 0x30;
}

int ADCA0(void)
```

```
{
	ADCA_CH0_MUXCTRL = 0x00;			//Set to Pin 0
	ADCA_CTRLA |= 0x04;					//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}

int ADCA1(void)
{
	ADCA_CH0_MUXCTRL = 0x08;			//Set to Pin 1
	ADCA_CTRLA |= 0x04;					//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}

int ADCA2(void)
{
	ADCA_CH0_MUXCTRL = 0x10;			//Set to Pin 2
	ADCA_CTRLA |= 0x04;					//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}

int ADCA3(void)
{
	ADCA_CH0_MUXCTRL = 0x18;			//Set to Pin 3
	ADCA_CTRLA |= 0x04;					//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}

int ADCA4(void)
{
	ADCA_CH0_MUXCTRL = 0x20;			//Set to Pin 4
	ADCA_CTRLA |= 0x04;					//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}

int ADCA5(void)
{
	ADCA_CH0_MUXCTRL = 0x28;			//Set to Pin 5
	ADCA_CTRLA |= 0x04;					//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}

int ADCA6(void)
{
	ADCA_CH0_MUXCTRL = 0x30;			//Set to Pin 6
	ADCA_CTRLA |= 0x04;					//Start Conversion on ADCA Channel 0
	while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);	//wait for conversion to complete
	delay_ms(5);
	int value = ADCA_CH0_RES;		//grab result
	return value;					//return result
}

int ADCA7(void)
```

```
{
        ADCA_CH0_MUXCTRL = 0x38;                    //Set to Pin 7
        ADCA_CTRLA |= 0x04;                                 //Start Conversion on ADCA Channel 0
        while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);    //wait for conversion to complete
        delay_ms(5);
        int value = ADCA_CH0_RES;               //grab result
        return value;                                       //return result
}
```

****

Header File

****

```
#ifndef  __PVR_h__
#define  __PVR_h__

#include <avr/io.h>
#include <avr/interrupt.h>
#include "PVR.h"


volatile int delaycnt;

void xmegaInit(void);

void delayInit(void);

void delay_ms(int cnt);

void delay_us(int cnt);

void lcd_command(unsigned char data);

void lcd_data(unsigned char data);

void lcd_init(void);

void lcd_write(char *str);

void lcd_clear(void);

void lcd_write_int(int value);

void ServoCInit(void);

void Servo_L_R(int Lvalue, int Rvalue);

void ServoC0(int value);

void ServoC3(int value);

void ServoC4(int value);

void ServoC5(int value);

void CMUsend(char *command);

char* CMUreceive(void);

void CMUTC(char *temp, int *x, int *y);

void ADCAInit(void);

int ADCA0(void);

int ADCA1(void);

int ADCA2(void);
```

```
int ADCA3(void);

int ADCA4(void);

int ADCA5(void);

int ADCA6(void);

int ADCA7(void);

#endif

**********************************************************************************
```