Final report

by Cheenu Madan

Robot name: Cuppy
Course: EEL5666 Intelligent Machines Design Laboratory
Instructors: Drs. Arroyo and Schwartz
TAs: Thomas Vermeer, Ryan Stevens, Mike Pridgen, Tim Martin, Devin Hughes

1. Opening

1.1 Abstract

Cuppy is a robot whose primary task is to collect red plastic cups. It uses the CMUcam2 and an IR rangefinder sensor to locate the cup and position the robot. It then uses its mechanical arm to pick up the cup and place the cup on a stand that it has.

1.2 Executive Summary

Cuppy consists of a square platform with a stand and an arm. The stand is a wooden strip mounted vertically on the board with a L-shaped bracket. The mechanical arm consists of a tilt drive, a gripper and a machinated ruler-shaped wooden strip used to join the tilt drive and the gripper. The tilt drive is capable of 150 degrees of rotation enabled by a Hitech HS645MG servo (which itself is capable of1 180 degrees of rotation). The gripper is capable of gripping cups of diameters 2 ½" inches and smaller.

A procedural software algorithm was written (see Appendix) for the process of picking up a plastic cup. The arm subsystem works smoothly and picks up a cup and deposits it on the stand smoothly.

Cuppy has three IR rangefinder sensors (Sharp GP2Y0A02YK0F), two which are used for obstacle avoidance and one of which is used for cup detection. These sensors are pretty reliable for obstacle avoidance. However, their inexact nature makes them unreliable for distance measurement. They have an empirically observed error range of ~2 inches. This contributes to the robot's poor positioning i.e. the robot is unable to position itself correctly at the required distance in front of the cup so that the arm can pick up the cup.

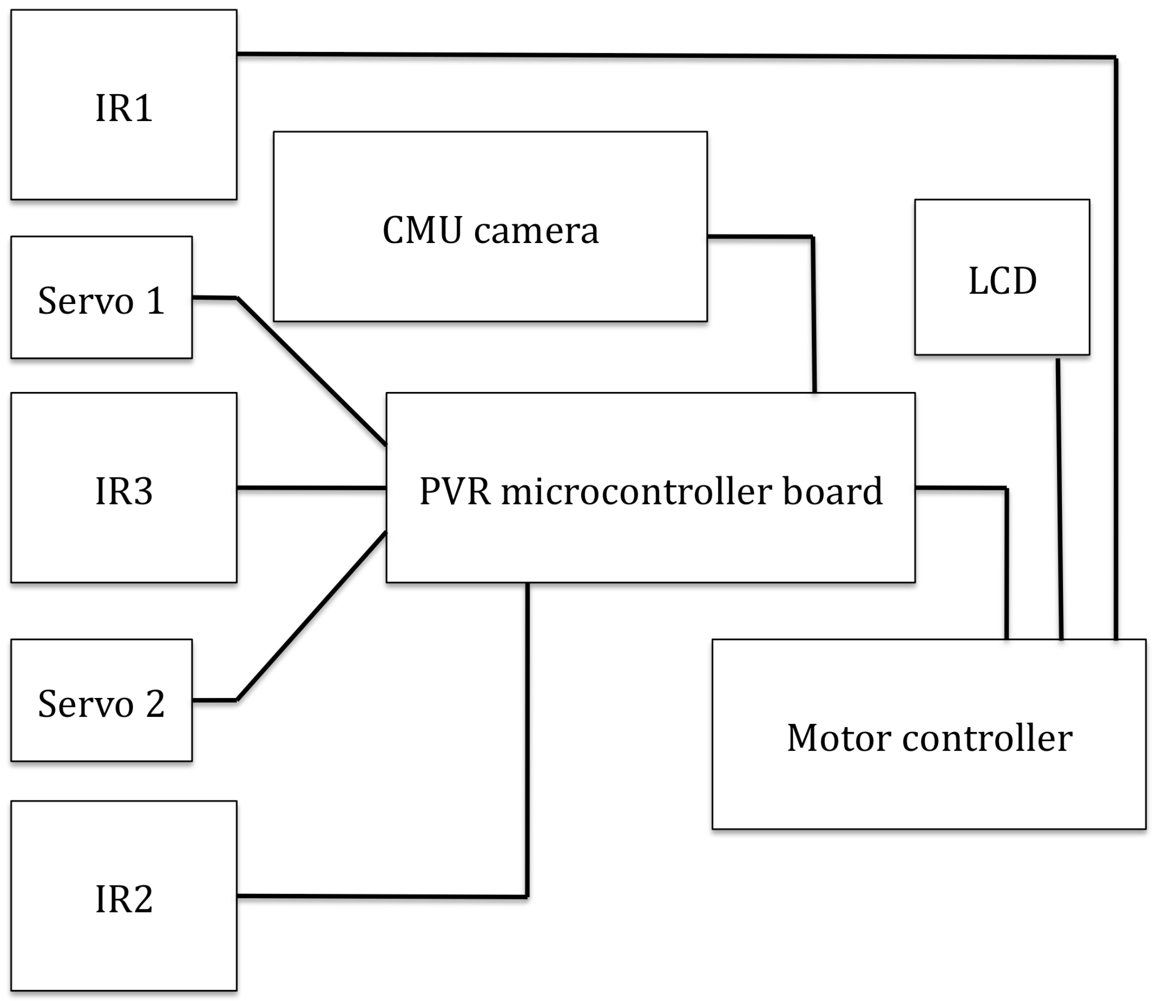
1.3 Introduction

The original objectives of the project was to build a autonomous robot that was capable of locating plastic cups, picking them up, stowing them and detecting when it cannot hold any more plastic cups.

2. Main body

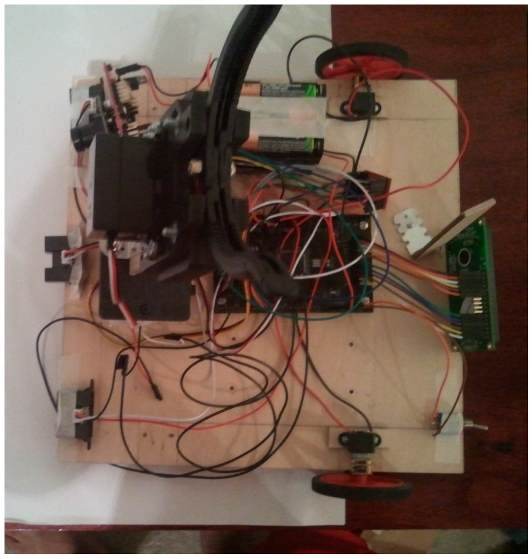
2.1 Integrated System

The block diagram below succinctly represents the integrated system.



IR1, IR2 – Obstacle avoidance
Servo 1 – Gripper
Servo 2 – Tilt drive
IR3 – Cup detection
CMU camera – Cup detection
Motor controller – For the motors.

2.2 Mobile platform



The mobile platform took a long time to be built. The gripper and the tilt drive had to be assembled and due to the inferior quality, all the parts weren't fitting smoothly. I had to use a cutting blade to modify the components so that they would fit smoothly.

I had to cut out the robot base, a stand, an ruler shaped rod for the arm and drill several holes in the robot base, stand and the rod so that various components could be mounted.

Originally my robot had four wheels and four motors but because each motors move at different speeds (although they shouldn't), the robot was not even moving forward properly. A modification to the robot design was made and a ball caster was purchased which replaced two of the wheels.

In retrospect, it would have been better if I had taken the two or three weeks to learn Solidworks and properly model the robot. I thought was robot platform was too simple to require modeling on Solidworks and having a TA cut out the parts on the T-Tech machine. But I ended up spending most of my time getting the platform to work and it now looks like an hacked together jungle of wires ☺.

2.3 Actuators

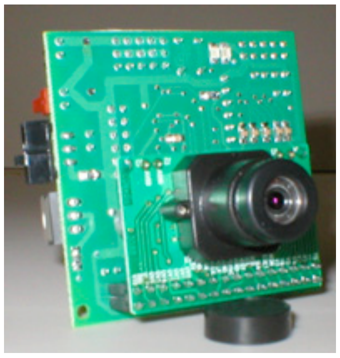
Mobile actuation was enabled using two 298:1 micro metal gear motors from Pololu. These gear motors are extremely small and so they do not take up a large footprint on the platform. They have the highest torque – lowest power ratings in their category, which met the overall goal of the robot not travelling too fast, and being able to support all the weight on the base.

Two servos (Hitech HS645) were the only two actuators used. One of the servos was used for the tilt drive and the other servo was used for the gripper.

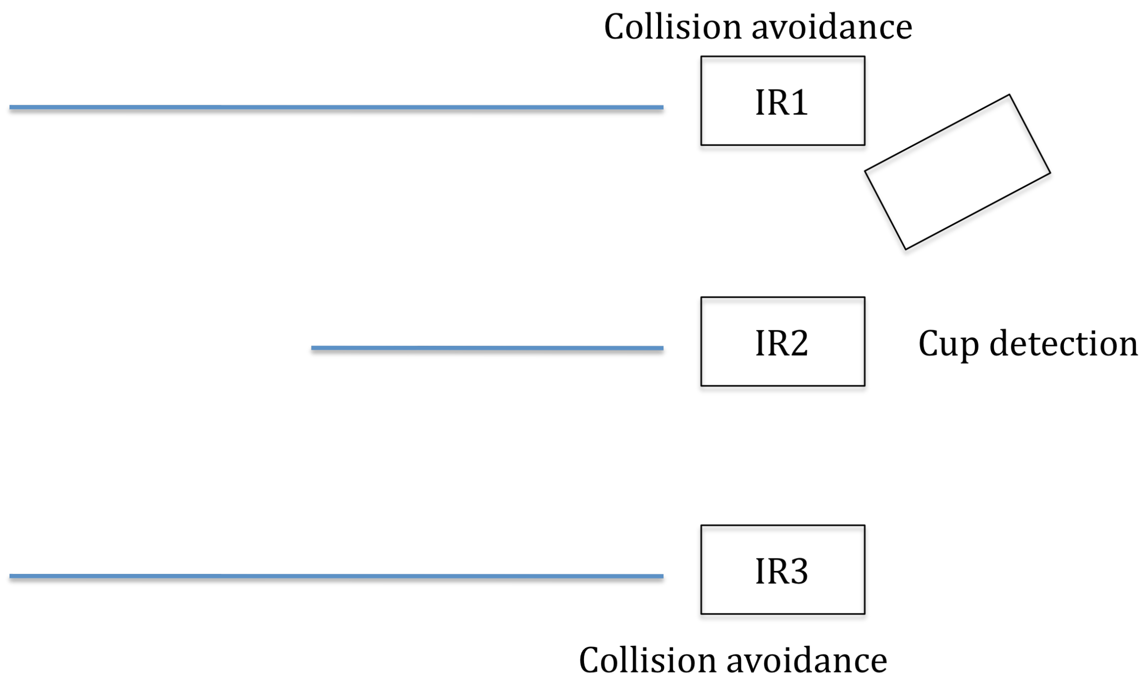
2.4 Sensors

Three Sharp GP2Y0A02YK0F IR rangefinder sensors are used. Two of them are used for obstacle avoidance and one of them is used for cup detection.

The CMUcam2 was intended to be the special sensor. However due to time constraints (I received the CMUcam just two weeks before media day), I was unable to integrate the CMUcam into the robot.



2.5 Behaviors



The software algorithm of the robot tracks the readings of the three IR sensors - IR1, IR2 and IR3. IR1 and IR3 are used for collision avoidance and IR2 is used for cup detection. The thresholds of IR1 and IR3 are intentionally set higher than of IR2 so that the robot does not go into obstacle avoidance mode when there is a cup in the vicinity.

Whenever the IR2 threshold is reached, then the robot initiates the arm to put up the cup. Since the IR2 sensor reaches its maximum value (=4095) at 10 to 12 inches away from the cup and the arm is 8 inches long, the robot much move forward at least 2 inches after detecting the cup which is incorporated in the software algorithm.

3. Conclusion

3.1 Conclusions

Being computer engineering major, I had little knowledge of the mechanical and electronics aspects. I tried to choose the platform design to be as simple as possible and go with the herd as far as the robot parts were concerned. In retrospect though, I underestimated just how much work building a platform and making sure all the components are integrated together electronically is. Initial days were spent in

apprehension of doing something wrong and frying up the expensive electronic components. The platform was reworked and reworked until I got it right.
I got little time to spend on the software aspect of the robot. In retrospect, it might have been actually better to get a pre-built robotics kit and then tried to focus on implementing some advanced programming behaviors. That will definitely be the approach I take with my second robot.

Regardless I'm really glad I took this course. I know now how to go about building a robot and if I could take the course over again, I would make a great robot! I would avoid the following mistakes

1. Not sticking to what you're good at (i.e. programming). Keep the electronics and mechanical aspects of the robot to a minimum.
2. Not modeling the entire platform down to the screws in Solidworks.
3. Also I would probably use a more powerful vision system (Webcam + Computer for processing) than onboard vision system like CMU/AVR camera.

3.2 Appendices

Program code

PVR.h

```
#ifndef  __PVR_h__
#define  __PVR_h__

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

#define LCD                  PORTK_OUT
#define LCDDDR         PORTK_DIR

volatile int delaycnt;

void xmegaInit(void);

void delayInit(void);

void delay_ms(int cnt);

void delay_us(int cnt);

void lcdDataWork(unsigned char c);

void lcdData(unsigned char c);

void lcdCharWork(unsigned char c);

void lcdChar(unsigned char c);

void lcdString(unsigned char ca[]);

void lcdInt(int value);

void lcdGoto(int row, int col);

void lcdInit(void);

void ServoCInit(void);

void ServoDInit(void);
void ServoC0(int value);

void ServoC1(int value);

void ServoC2(int value);

void ServoC3(int value);

void ServoC4(int value);

void ServoC5(int value);

void ServoD0(int value);

void ServoD1(int value);

void ServoD2(int value);

void ServoD3(int value);

void ServoD4(int value);

void ServoD5(int value);

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);

void MotorInit ();

void MotorDrive (int speed, int mode);
void CMUsend (char *command);

char CMUreceive (void);

#endif
```

<u>PVR.c</u>

```
#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 *
 *******/

#define LCD              PORTK_OUT
#define LCDDDR         PORTK_DIR

void lcdDataWork(unsigned char c)
{
    c &= 0xF0;                                //keep data bits, clear the rest
    c |= 0x08;                               //set E high
    LCD = c;                                //write to LCD
    delay_ms(2);                           //delay
    c ^= 0x08;                                //set E low
    LCD = c;                                //write to LCD
    delay_ms(2);                           //delay
    c |= 0x08;                              //set E high
    LCD = c;                                //write to LCD
    delay_ms(2);                            //delay
}

void lcdData(unsigned char c)
{
    unsigned char cHi = c & 0xF0;           //give cHi the high 4 bits of c
    unsigned char cLo = c & 0x0F;           //give cLo the low 4 bits of c
    cLo = cLo * 0x10;                       //shift cLo left 4 bits
    lcdDataWork(cHi);
    lcdDataWork(cLo);
}

void lcdCharWork(unsigned char c)
{
    c &= 0xF0;                              //keep data bits, clear the rest
    c |= 0x0A;                              //set E and RS high
    LCD = c;                                //write to LCD
    delay_ms(2);                            //delay
    c ^= 0x08;                              //set E low
    LCD = c;                                //write to LCD
    delay_ms(2);                            //delay
    c |= 0x08;                              //set E high
    LCD = c;                                //write to LCD
    delay_ms(2);                            //delay
}

void lcdChar(unsigned char c)
{
    unsigned char cHi = c & 0xF0;           //give cHi the high 4 bits of c
    unsigned char cLo = c & 0x0F;           //give cLo the low 4 bits of c
    cLo = cLo * 0x10;                       //shift cLo left 4 bits
    lcdCharWork(cHi);
    lcdCharWork(cLo);
}


void lcdString(unsigned char ca[])
{
    int i = 0;
    while (ca[i] != '\0')
    {
        lcdChar(ca[i++]);
    }
}
```

```
void lcdInt(int value)
{
    int temp_val;
    int x = 10000;
    int leftZeros=5;

    if (value<0)
    {
       lcdChar('-');
          value *= -1;
    }

  while (value / x == 0)
  {
         x/=10;
         leftZeros--;
  }


  while ((value > 0) || (leftZeros>0))
  {
    temp_val = value / x;
    value -= temp_val * x;
    lcdChar(temp_val+ 0x30);
    x /= 10;
    leftZeros--;
  }

  while (leftZeros>0)
    {
         lcdChar(0+ 0x30);
    leftZeros--;
  }

  return;
}

void lcdGoto(int row, int col)
{
    unsigned char pos;
    if ((col >= 0 && col <= 19) && (row >= 0 && row <= 3))
    {
         pos = col;
         if (row == 1)
              pos += 0x40;
```

```
        else if (row == 2)
            pos += 0x14;
        else if (row == 3)
            pos += 0x54;
        lcdData(0x80 + pos);
    }
}

void lcdInit(void)
{
    delayInit();                    //set up the delay functions
    LCDDDR = 0xFF;                      //set LCD port to outputs.
    delay_ms(20);                     //wait to ensure LCD powered up
    lcdDataWork(0x30);                    //put in 4 bit mode, part 1
    delay_ms(10);                     //wait for lcd to finish
    lcdDataWork(0x30);                    //put in 4 bit mode, part 2
    delay_ms(2);                     //wait for lcd to finish
    lcdData(0x32);                    //put in 4 bit mode, part 3
    lcdData(0x2C);                    //enable 2 line mode
    lcdData(0x0C);                    //turn everything on
    lcdData(0x01);                    //clear LCD
}

/*********
 * 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
}

void ServoDInit(void)
{
    TCD0_CTRLA = 0x05;                          //set TCC0_CLK to CLK/64
    TCD0_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
    TCD0_PER = 10000;                           //20ms / (1/(32MHz/64)) = 10000.  PER =
Top
    TCD1_CTRLA = 0x05;                          //set TCC1_CLK to CLK/64
    TCD1_CTRLB = 0x33;                          //Enable OC A and B.  Set to Single Slope
PWM
                                            //OCnX = 1 from Bottom to CCx and 0 from CCx
to Top
    TCD1_PER = 10000;                           //20ms / (1/(32MHz/64)) = 10000.  PER =
Top
    PORTD_DIR = 0x3F;                           //set PORTC5:0 to output
    TCD0_CCA = 0;                           //PWMC0 off
    TCD0_CCB = 0;                           //PWMC1 off
    TCD0_CCC = 0;                           //PWMC2 off
    TCD0_CCD = 0;                           //PWMC3 off
    TCD1_CCA = 0;                           //PWMC4 off
    TCD1_CCB = 0;                           //PWMC5 off
}

void ServoC0(int value)
{
    if (value > 100)                        //cap at +/- 100
        value = 100;                        //  -100 => 1ms
    else if (value < -100)                 //  0    => 1.5ms
        value = -100;                       //  100  => 2ms
    value *= 5;                                 //multiply value by 2.5
    value /= 2;                                 //  new range +/- 250
    TCC0_CCA = (750 + value);                   //Generate PWM.
}

void ServoC1(int value)
{
    if (value > 100)                        //cap at +/- 100
        value = 100;                        //  -100 => 1ms
    else if (value < -100)              //  0    => 1.5ms
        value = -100;                    //  100  => 2ms
    value *= 5;                            //multiply value by 2.5
    value /= 2;                            //  new range +/- 250
    TCC0_CCB = (750 + value);              //Generate PWM.
}

void ServoC2(int value)
{
    if (value > 100)                      //cap at +/- 100
        value = 100;                     //  -100 => 1ms
    else if (value < -100)              //  0    => 1.5ms
        value = -100;                    //  100  => 2ms
    value *= 5;                            //multiply value by 2.5
    value /= 2;                            //  new range +/- 250
    TCC0_CCC = (750 + value);              //Generate PWM.
}

void ServoC3(int value)
{
    if (value > 100)                      //cap at +/- 100
        value = 100;                     //  -100 => 1ms
    else if (value < -100)              //  0    => 1.5ms
        value = -100;                    //  100  => 2ms
    value *= 5;                            //multiply value by 2.5
    value /= 2;                            //  new range +/- 250
    TCC0_CCD = (750 + value);              //Generate PWM.
}

void ServoC4(int value)
{
    if (value > 100)                      //cap at +/- 100
        value = 100;                     //  -100 => 1ms
    else if (value < -100)              //  0    => 1.5ms
        value = -100;                    //  100  => 2ms
    value *= 5;                            //multiply value by 2.5
    value /= 2;                            //  new range +/- 250
    TCC1_CCA = (750 + value);              //Generate PWM.
}

void ServoC5(int value)
{
    if (value > 100)                      //cap at +/- 100
        value = 100;                     //  -100 => 1ms
    else if (value < -100)              //  0    => 1.5ms
        value = -100;                    //  100  => 2ms
    value *= 5;                          //multiply value by 2.5
    value /= 2;                          //  new range +/- 250
    TCC1_CCB = (750 + value);            //Generate PWM.
}

void ServoD0(int value)
{
    if (value > 100)                     //cap at +/- 100
        value = 100;                   //  -100 => 1ms
    else if (value < -100)             //  0    => 1.5ms
        value = -100;                  //  100  => 2ms
    value *= 5;                          //multiply value by 2.5
    value /= 2;                          //  new range +/- 250
    TCD0_CCA = (750 + value);            //Generate PWM.
}

void ServoD1(int value)
{
    if (value > 100)                     //cap at +/- 100
        value = 100;                   //  -100 => 1ms
    else if (value < -100)             //  0    => 1.5ms
        value = -100;                  //  100  => 2ms
    value *= 5;                          //multiply value by 2.5
    value /= 2;                          //  new range +/- 250
    TCD0_CCB = (750 + value);            //Generate PWM.
}

void ServoD2(int value)
{
    if (value > 100)                     //cap at +/- 100
        value = 100;                   //  -100 => 1ms
    else if (value < -100)             //  0    => 1.5ms
        value = -100;                  //  100  => 2ms
    value *= 5;                          //multiply value by 2.5
    value /= 2;                          //  new range +/- 250
    TCD0_CCC = (750 + value);            //Generate PWM.
}

void ServoD3(int value)
{
    if (value > 100)                     //cap at +/- 100
        value = 100;                   //  -100 => 1ms
    else if (value < -100)             //  0    => 1.5ms
        value = -100;                  //  100  => 2ms
    value *= 5;                          //multiply value by 2.5
    value /= 2;                          //  new range +/- 250
    TCD0_CCD = (750 + value);              //Generate PWM.
}

void ServoD4(int value)
{
    if (value > 100)                          //cap at +/- 100
        value = 100;                         //  -100 => 1ms
    else if (value < -200)                 //  0    => 1.5ms
        value = -200;                        //  100  => 2ms
    value *= 5;                                //multiply value by 2.5
    value /= 2;                                //  new range +/- 250
    TCD1_CCA = (750 + value);              //Generate PWM.
}

void ServoD5(int value)
{
    if (value > 100)                          //cap at +/- 100
        value = 100;                         //  -100 => 1ms
    else if (value < -100)                 //  0    => 1.5ms
        value = -100;                        //  100  => 2ms
    value *= 5;                                //multiply value by 2.5
    value /= 2;                                //  new range +/- 250
    TCD1_CCB = (750 + value);              //Generate PWM.
}

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

void ADCAInit(void)
{
    delayInit();
    ADCA_CTRLB = 0x00;                        //12bit, right adjusted
    ADCA_REFCTRL = 0x10;                 //set to Vref = Vcc/1.6 = 2.0V (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;               //set the speed to slow for higher accuracy
    ADCA_CTRLA |= 0x01;                     //Enable ADCA
}

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
}

void MotorInit ()
{
    // Set F0 and F1 to output
    // Set J0, J1, J2, J3, J4 to output
    PORTF_DIR = 0x03;
    PORTJ_DIR = 0x1F;
    // Initialize outputs to L
    PORTF_OUT = 0x00;
    PORTJ_OUT = 0x00;
}

void MotorDrive (int speed, int mode)
{

    if (mode == 1)
    { // Rotate counter-clockwise
        // PWNA - F0 -> 1
        // PWNB - F1 -> 1
        // F0 and F1 to H
        PORTF_OUT = 0x03;
        // AIN2 -> J0 -> 0
        // AIN1 -> J1 -> 1
        // STBY -> J2 -> 1
        // BIN2 -> J3 -> 0
        // BIN1 -> J4 -> 1
        PORTJ_OUT = 0x16;
    }

    if (mode == 0)
    { // Rotate clockwise
        // PWNA - F0 -> 1
        // PWNB - F1 -> 1
        PORTF_OUT = 0x03;
        // AIN2 -> J0 -> 1
        // AIN1 -> J1 -> 0
        // STBY -> J2 -> 1
        // BIN2 -> J3 -> 1
        // BIN1 -> J4 -> 0
        PORTJ_OUT = 0x0d;
    }

    if (mode == 2)
    { // Stop
        // PWNA - F0 -> 1
        // PWNB - F1 -> 1
        PORTF_OUT = 0x03;
        // AIN2 -> J0 -> 0
        // AIN1 -> J1 -> 0
        // STBY -> J2 -> 1
        // BIN2 -> J3 -> 0
        // BIN1 -> J4 -> 0
        PORTJ_OUT = 0x04;
    }
    if (mode == 3)
    { // Turn
        // PWNA - F0 -> 1
        // PWNB - F1 -> 1
        PORTF_OUT = 0x03;
        // AIN2 -> J0 -> 0
        // AIN1 -> J1 -> 1
        // STBY -> J2 -> 1
        // BIN2 -> J3 -> 0
        // BIN1 -> J4 -> 0
        PORTJ_OUT = 0x06;
    }
}

void CMUsend(char *command) {
    int i = 0;
    while (command[i] != '\0') {
        //While command does not end do...
        // Data Register Empty Flag: check if data register is empty
        while (!(USARTE0_STATUS & (1<<USART_DREIF_bp)));
        //USARTE0_DATA is shared by the transmit and receive
        USARTE0_DATA = command[i];
        //if data is sent TXCIF is set
        while (!(USARTE0_STATUS & (1<<USART_TXCIF_bp)));
            i++;

    }
}

char CMUreceive(void) {
    int i=0;
    static char *data;
    do {
        // wait for receive is complete
        while (!(USARTE0_STATUS & (1<<USART_RXCIF_bp))){}
        //Put data into string
        data[i] = USARTE0_DATA;
    } while (data[i++] != '\r'); //Since all trasnfers are ended by \r
wait for it to happen
    return data;
}
```

<u>Cuppy.c</u>

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

```
//#define THRESHOLD 2500 //At home //Obstacle avoidance
#define THRESHOLD 3200 //Lab corridor //Obstacle avoidance
#define CUPTHRESHOLD 4094 //Lab corridor //Cup detection

void main(void)
{
    xmegaInit();                              //setup XMega
    delayInit();                             //setup delay functions

    // ************************************
    //Final
    // ************************************
    //ServoD5 - Direct Drive Tilt Motor Servo
    //ServoD4 - Gripper Servo
    //ADCA0 - Left IR sensor
    //ADCA1 - Right IR sensor
    //ADCA4 - Central IR sensor
    // ************************************

    ADCAInit ();
    lcdInit ();
    MotorInit ();
    ServoDInit();

    int reading_left;
    int reading_right;
    int reading_cup;

    ServoD5 (100); // Open arm fully wide
    ServoD4 (0); // Move tilt motor to initial position
    delay_ms (2000); // Wait a bit...

    while (1) {

        // Start moving
        MotorDrive (100,0);

        // Assign values to variable for later clarity in code.
        reading_cup = ADCA4 ();
        reading_left = ADCA0 ();
        reading_right = ADCA1 ();

        // Display information on LCD for debugging
        // purposes.
        lcdGoto (0,0);
        lcdInt (reading_left);
        lcdGoto (1,0);
        lcdInt (reading_right);
        lcdGoto (1,5);
        lcdInt (reading_cup);

        // Both the left and right sensors have high reading values.
        // We are defintely approaching an obstacle like a wall.
        if (reading_left > THRESHOLD || reading_right > THRESHOLD) {
            MotorDrive (100,2); //Stop
            delay_ms (1000);
            MotorDrive (100,3); //U-turn
            delay_ms (2500);
        }
        // Found cup
        else if (reading_cup > CUPTHRESHOLD) {
            MotorDrive (100,2); // Stop
            delay_ms (5000);
            ServoD4 (-200);     // Bring arm down
            delay_ms (5000);
            MotorDrive (100,0); // Move
            delay_ms (400);    // for 2 seconds
            MotorDrive (100,2); // Stop
            delay_ms (5000);
            ServoD5 (-70);     // Start gripping
            delay_ms (5000);
            ServoD5 (-100);     // Fully grip
            delay_ms (5000);
            ServoD4 (100);     // Move arm above stand
            delay_ms (5000);
            ServoD5 (100);     // Release grip
            delay_ms (5000);
            ServoD4 (0);       // Move back to initial position
        }
    }

// End of final

}
```