University of Florida

Department of Electrical and Computer Engineering

EEL5666

Intelligent Machine Design Lab

SHOTBOT

Final Report

Aaron Chinault

Table of Contents

Abstract

Shotbot is an autonomous bartending robot. It approaches patrons, receives and validates payment/proof of age, and then gives them one of 20 shots selected by the user from an internal drink database. The robot is a 2 wheel(and 1 castor) design that distributes the liquid from a partitioned tank with 6 12VDC pumps.

Executive Summary

The goal of Shotbot is to be a fully functional robot bartender, capable of doing all of the major customer related tasks a normal bartender must do. The includes approaching(actually passing by) patrons at a bar, collecting their order, payment for order, verification of their age, and finally mixing and giving them their order. It has a simple, yet sleek, chassis design and moves around on a two wheel and one castor system. It begins its program by following the edge of a bar using an innovative 3 IR sensor system. While doing this it avoids obstacles with its forward looking ultrasonic range finder and bump switch (technically it does not avoid the obstacle, it senses it and complains.) When a patron taps it on a shoulder switch it will then activiate the LCD backlights, turn to the patron, and request ID verification. The verification is accomplished by scanning a barcode located on a wristband. If validation fails (patron tries to defraud Shotbot) a loud alarm will sound. Otherwise, Shotbot will request a token and then validate that tokens authenticity (alarm sounds if fraud). With payment and id verification completed Shotbot collects the patron's order from a series of buttons next to the LCD displays and then dispenses the shot using six PWM controlled pumps. Almost all of these operations are controlled through a single multi-headed I2C bus by using many independent modules.

Introduction

The Shotbot is a robot that is born out of a need for better and cheaper bartending services (emphasis on various shots.) As a robot this bartender does not have the same problems that human bartenders have. It never over pores, calls in sick, goes on vacation, or gets paid. Establishment owners often spend hundreds of dollars on pour accuracy testing for their tenders. The money saved from this alone could cover Shotbot's purchase.

In addition to the owners, patrons will also love Shotbot to as they can get their drinks cheaper because they do not have to worry about tips. There is also just something inherently nifty in getting a drink dispensed from an aesthetically pleasing robot with all sorts of moving things going on. This means that owning a Shotbot will act as a draw for new patrons that just want to experience the goodness that is Shotbot and thus leading to even more profit. This paper will go into detail on the various functions and features of Shotbot including its sensors, actuators, behaviors, and more.

Integrated System

Shotbot consists of a 2 wheeled moving platform that has a shot dispensing system built into it. This platform will be able to move across a bar area using its ultrasonic range finder and bump sensors for object avoidance and infrared detectors for detecting the edge of the bar. When hailed by a patron, it will request a scanning of an ID band and a drink token from the patron. Upon validation of the wristband barcode and insertion of token, Shotbot will validate the authenticity via a multiple red led/CDS validator system. If either of these validation procedures fails, an alarm will sound. Once token authenticity has been confirmed Shotbot will ask the user what kind of shot they want using its 6 LCD screens. The patron then selects his/her drink and the Shotbot dispensing mechanism goes into action. A holder tray rotates out and a shot glass is then placed in the holder by the patron (shot glasses are located on the top of Shotbot). Then the tray rotates back into the platform the appropriate pumps turn on, dispensing until the entire shot is made. Then the shot glass rotates out and the patron can imbibe it. The Shotbot will then shut down(can be made continuous with minor code change, but currently it shuts off to reserve battery power).

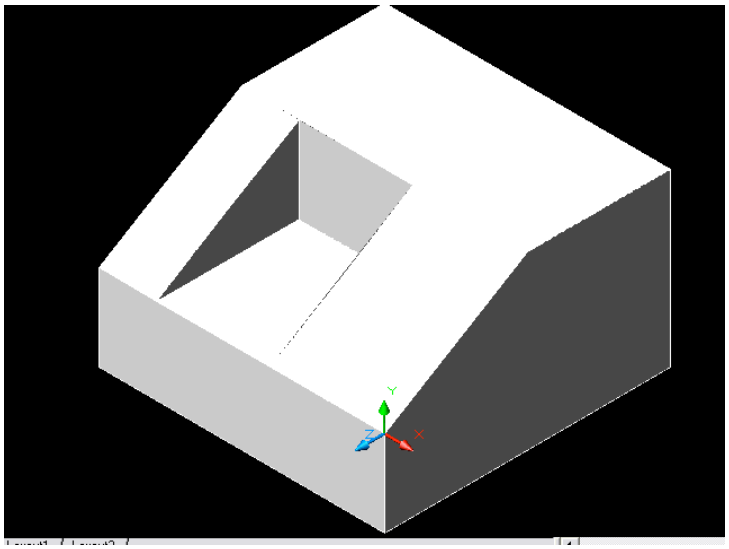
In order to perform these tasks an AVRMega128 chip running at 16MHz controls all of the sensors and behaviors. It is mounted on an Olimex board and has 54 I/O pins which when combined with its hardware I2C (TWI) bus provides ample control pins of all the applications. In order to assist the Mega128 a SD20 external servo controller is used for controlling the servos and an external I2C eeprom is used to hold the recipes.

Integration system

` Shotbot is controlled by a Universal Robot Control Bus(UCRB). This bus consists of 5 wires and control/power almost the entirety of Shotbot (The exception being the barcode scanner.) The wires are the two I2C lines and 2 independently regulated 5V supplies and a common ground. More info on this system can be found on the attached special sensor report.

Mobile Platform

The platform is roughly 15” by 15” by 10” and consists of an outer shell and an inner cavity where all the electronics and mechanisms are housed. It uses two quarter scale servos with attached wheels for locomotion along with a rear metal ball castor for support. Upon the larger sloping surface all of the LCDs are mounted along with 2 rows of buttons. The Thinner sloped surface holds the coin validator and ID scanner. In the recess the shot is distributed.



Conceptual picture of Shotbot’s chassis(minus components)

Inside the back of the Shotbot lies the heart of the dispersion mechanism, a 14”X2”X6” 6 section acrylic tank with each tank having its own pump. This system holds and provides the distributing power for all the liquors. Tubes from these pumps run up to the recess area where a dropped shotglass can be filled.

Actuation

A. Objectives

The objectives of the actuators are to control the locomotion of the platform as well as the internal dispensing mechanism.

B. Types of Actuation

12VDC pumps- Originally used on Butlerbot, these are surplus windshield wiper pumps with a _" intake and 1/8" outtake, no check valves.

Only servos are used for creating movement but they have been rotary adapted by cutting the internal stops and setting the internal pot the servos can be hacked to run like a standard motors(but with easier control.)

C. Applications

a. Dispensing the liquid(6 pumps)

b. 2.5" Wheels (2 rotary adaptation quarter scale)

c. Rotating shot glass tray (1 standard)

D. Characteristics

HS-311 Standard Servo	42 oz-in of torque	.17 sec/60 deg no load
HS-700BB Quarter Scale	170 oz-in of torque	.22 sec /60 deg no load

Sensors

A. Objective

The objective of the sensors is to relay the status of the world to the robot. This allows the robot to perform obstacle avoidance/eow detection and receive requests from users.

B. Sensors Used

a. Bump Sensors

The bump sensors are old keyboard switches that are used to detect if the front of the robot collides with an obstacle. There are 3 placed on the front of the robot. Two are placed on the bottom of the front of the robot (about 4 inched in from either side). These switches have a bar running between them and around the entire front of the robot to allow for complete object detection.

b. SF08 Ultrasonic Range Finder

The SF08 is an I2C bus operated Ultrasonic range finder. It pings every 65ms and stores the calculated ranging inside its internal registers. The mega128 then reads the registers and can use this ranging information to engage in obstacle avoidance. In order to improve the field of view of the SF08 it is mounted upon a HS-311 servo and thus allowing it to rotate up to 180 deg.

c. Hamamatsu P5587 IR Photoreflector

These sensors give off a digital signal (high) when there is no object detected within and inch or so of them (depending of the object color). Three of these sensors are utilized in the edge of bar following behavior.

d. 2x Ultra bright red led and CDS cell(token validator)

The LEDs is collimated and pointed at the CDS cell within the closed environment of the token validation system. They are placed in a + configuration so that one CDS can verify longitudinally(along the edge of the coin) and the other can verify widthwise. Upon a token(or fake token) striking the bottom of the mechanism longitudinal the CDS cell will generate a reading of the light. This serves to notify the system that a longitudinally opaque object has fallen into the validator(presumably a token). Since the tokens are created out of tinted transparent acrylic with a black border a certain fraction of the light will shine through the width unto the CDS cell. The black border, however is opaque and allows the tripping of the longitudinal sensor. By measuring the resistance of the width detecting cell it can be determined if the inserted token is of the proper opacity and color, thereby validating its authenticity.

e. Push Button Array

The push button array is made out of 11 simple momentary push buttons(keyboard keys with custom _" square caps). The array is used to

allow the user to select the various menu options displayed upon the LCDs. These buttons are connected to a decoder circuit that converts the 11 lines into a total of 2 bytes that can be read off of the I2C bus. Each of the 11 switches will have a code and there will be a distress code(if two buttons get triggered at the same time) and an empty code.

Behaviors

A. Objectives

The purposes of the behaviors is to control the robot and make it do all of the required functions.

B. Behaviors

a. Ping- This behavior takes care of the panning and ranging form the sonar
b. Edge of bar following- Uses inputs from the IR sensors to keep the robot on the edge bartop. It does this by adjusting the speed of the wheels to keep 2 IR sensors parallelll with the edge and using a 3rd sensor to detect when there is a 90 degree turn to be made.
c. ID verification- Scans a barcode and validates it.
d. Token Validation- Waits for the token and validates it. It then sets up the appropriate next response.
e. Drink Selection- Presents the options to the user and awaits his/her choice of shot.
f. Drink Dispension- Sends instructions to create the shot according to the recipe found in the database.

Module Listing

LCD Pro 9000- This controller controls the 6 LCDs using the I2C line. It uses two I2C port expanders to convert the I2C data to parallel and adjust which enables are turned on thereby changing which LCD clocks in the data. A power fet is also on the board to allow software control of the backlights.

Switch Decoder Module- The decoder takes in the 11 switches of the switch array and decodes their single onto the I2C bus using two port expanders.

Power Module- Holds all seven of the power regulators(2 for TTL and one for high current applications such as servos and backlights) and holds the pins of the UCRB, housing the I2C pull-up resistors and a bus of pins allowing other mnodules to hook together.

Servo Controller/ EEPROM modules- Contains the SD20 servo controller and I2C EEPROM, with associated hardware.

Alarm Module- Contains a port expander and a BJT to control the alarm.

CPU Module- Holds the Mega128 and the scanner interface.

IR/Bump Module- Contains inputs for the IR sensors and bump switches via a port expander.

Sonar Module- Performs the ultrasonic pinging at the front of the robot.

TTS Module- Allows the robot to speak(albeit softly at the moment).

Pump Controller- Consists of a port expander, Hex inverter, and six power fets. These fets are PWMd at a 25% duty cycle when the software wants to turn one of them on. At this duty cycle a shot glass is filled in about 3 seconds. The inverters are used to make sure that the initial value on the port of 0xFF does not cause all of the pumps to turn on briefly on boot up, thereby causing a massive current spike(10amps+) and squirting liquid all over the electronics(bad times).

Experimental Layout and Results

Not really applicable, the sensors are relatively straight forward and no sizeable experiments had to be made.

Conclusion

In the end Shotbot met and exceeded many of my initial goals. The only systems that I did not get operational were the shot glass dropper and the panning rangefinder. I ruled out the glass dropper because I was not able to economically obtain materials that would guarantee me a successful drop every time, and I was not willing to take the risk of a botched pour, spilling an oz. of liquid into my robot's interior. The range finder does not pan because I was not able to get a new sprocket fast enough to implement it, although the mechanics of the system are in place and you can niftily swivel the rangefinder with your finger. Other optional features, such as the token validator and ID checker were able to be flawlessly implemented. Overall I think my robot turned out extremely well and there is little if anything I would change If I had to build it again given the same budgetary restraints (it would have been nice to have my modules sent out for PCBing though.) In the future a interactive barmaster interface could be added with little effort t(scanner operates off of keyboard scan codes.) This would allow dynamic changing of the drink database and barcode, as well as many diagnostic features.

Documentation

Servos

www.servocity.com

Dev Board

www.sparkfun.com

Wheels, IR sensors, TTS module, and SF08

www.junun.org

Various parts

www.radioshack.com

Pumps

www.allelectronics.com

Appendices

Here is a listing of the complete code:

```
#include <mega128.h>


#asm
   .equ __i2c_port=0x12
   .equ __sda_bit=1
   .equ __scl_bit=0
#endasm
#include <i2c.h>
#include <delay.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>


#define SONAR_ADDR 0xE0

#define TTS_ADDR 0xC4
#define TTS_CMD_REGISTER 0
#define NOP 0
#define TTS_NULL 0
#define SPKBUF 0x40

#define SERVO_ADDR 0xC2

#define EEPROM_ADDR 0b10100000

#define ALARM_ADDR 0b01001110

#define LCD_DATA_ADDR 0b01000000
#define LCD_CTRL_ADDR 0b01000010
#define ON  1
#define OFF 0

#define SWITCH_ADDR_LOW  0b01000100
#define SWITCH_ADDR_HIGH 0b01000110

#define TOKEN_A2D_ADDR   0b10010000
```

```
#define DRINK_NAME  0
#define SPOKEN_NAME 20
#define VODKA_AMT 50
#define BLUE_AMT 51
#define MIDORI_AMT 52
#define PEACH_AMT 53
#define CRAN_AMT 54
#define OJ_AMT 55

#define EOB_ADDR 0b01001100

#define LEFT_MOTOR 1
#define RIGHT_MOTOR 2

#define FALLING_EDGE 0x20
#define RISING_EDGE 0x30

#define PUMP_ADDR    0b01001000

#define TURNTABLE_SERVO  4

// TES STUFF
#define BUFF_SIZE 64
unsigned char edge, bitcount;// 0 = neg. 1 = pos.
unsigned char kb_buffer[BUFF_SIZE];
unsigned char *inpt, *outpt;
unsigned char buffcnt;

//TES

typedef unsigned char byte;


  int DR=0b00000000, CR=0b00000000;

  int TTS_Volume=0, TTS_Pitch=0, TTS_Speed=7;
  int Ping_Dist=160;

  int Speech_Holder=0;

  int temp=1, hold=0;
  int Switchtemp;
  int Switch_Status_low=0,Switch_Status_high=0;

  int a2d_high,a2d_low;
```

```
  int drindex=0,mod=0,selection=999;

  int IR_Front=0, IR_Right=0, IR_Left=0, Front_Bump=0, Top_Bump=0;
  int Readings;

 int Scanned_Code[12]={0},Code_Ready=0,
Correct_Code[12]={6,6,9,9,6,2,3,1,0,5,7,4};
int Valid_ID=1;

flash byte char0[8]={
0b1101110,
0b1110001,
0b1111111,
0b1110001,
0b1110001,
0b1110001,
0b1101110,

0b1000000};

flash byte char1[8]={
0b1101110,
0b1110001,
0b1111111,
0b1110001,
0b1111111,
0b1111111,
0b1101110,

0b1000000};




/*****************************************************************************
**********************
************************
************************
************************          I2C LCD FUNCTIONS
************************
************************
************************
*****************************************************************************
*********************/
```

```
// Updates the command reg I2C port
void Write_CR()
{
i2c_start();
i2c_write(LCD_CTRL_ADDR);
i2c_write(CR);
i2c_stop();
}

// Updates the data reg I2C port
void Write_DR()
{
i2c_start();
i2c_write(LCD_DATA_ADDR);
i2c_write(DR);
i2c_stop();
}

// Selectivly toggles the enable pin of the desired lcd(latch in)
void Toggle_E(int tognum)
{
Write_DR();

if(tognum==1)
CR=CR | 0b00000100;
if(tognum==2)
CR=CR | 0b00001000;
if(tognum==3)
CR=CR | 0b00010000;
if(tognum==4)
CR=CR | 0b00100000;
if(tognum==5)
CR=CR | 0b01000000;
if(tognum==6)
CR=CR | 0b10000000;
if(tognum==0)
CR=CR | 0b11111100;

Write_CR();
delay_ms(1);

if(tognum==1)
CR=CR & 0b11111011;
if(tognum==2)
CR=CR & 0b11110111;
if(tognum==3)
```

```
CR=CR & 0b11101111;
if(tognum==4)
CR=CR & 0b11011111;
if(tognum==5)
CR=CR & 0b10111111;
if(tognum==6)
CR=CR & 0b01111111;
if(tognum==0)
CR=CR & 0b00000011;

Write_CR();
}

// Sets the register select bit in the CR to the data position
void RS_DATA()
{
CR=CR | 0b00000001;
}

// Sets the register select bit in the CR to the command position
void RS_CMD()
{
CR=CR & 0b11111110;
}

// Sends the initialization sequence to all 6 LCDs
void lcd_init()
{
   DR=0b00110000;
  RS_CMD();
  delay_ms(15);
  Toggle_E(0);
  delay_ms(5);
  Toggle_E(0);
  delay_ms(1);
  Toggle_E(0);

  delay_ms(5);
  DR=0b00111111;
  Toggle_E(0);

  delay_ms(2);
  DR=8;
  Toggle_E(0);

  delay_ms(2);
```

```
  DR=1;
  Toggle_E(0);

  delay_ms(2);
  DR=6;
  Toggle_E(0);

  delay_ms(2);
  DR=12;
  Toggle_E(0);

}

// Outputs a string on selected LCD
void lcd_putsf(int num,char flash *str)
{
int n=0;                                    // Counter

     RS_DATA();
        while(n<strlenf(str))         // Transfer individual atoms of the string
                {
                DR=str[n];
                n++;
                delay_us(100);
                Toggle_E(num);
                }
}

// Clears selected lcd
void lcd_clear(int num)
{
RS_CMD();

DR=1;
delay_ms(1);
Toggle_E(num);
}

// Sets the LCD backlights either ON or OFF
void lcd_backlight(int onoff)
{
if (onoff==ON)
CR=CR | 0b0000010;
else
CR=CR & 0b1111101;
Write_CR();
```

```
}

// Sets the cursor on selected screen to x,y
void lcd_gotoxy(int num,int x, int y)
{
  RS_CMD();
  delay_ms(1);
  DR=0b10000000+(0x40*y)+x;
  Toggle_E(num);
}

// Places a character ont he selected LCD
void lcd_putchar(int num, int value)
{
RS_DATA();
DR=value;
delay_ms(1);
Toggle_E(num);
}

// Places a 1-5 digit integer on the selected display
void lcd_putint(int num, int int_value)
{
int temp_value;
temp_value=int_value;
if(int_value>9999)
{ lcd_putchar(num, temp_value/10000+0x30); temp_value=temp_value%10000;}
if(int_value>999)
{ lcd_putchar(num, temp_value/1000+0x30); temp_value=temp_value%1000;}
if(int_value>99)
{ lcd_putchar(num, temp_value/100+0x30); temp_value=temp_value%100;}
if(int_value>9)
{ lcd_putchar(num, temp_value/10+0x30); temp_value=temp_value%10;}
lcd_putchar(num, temp_value+0x30);
}

// Defines a custom character given by a byte array at selected CGRAM addr
void define_char(byte flash *pc,byte char_code)
{
byte i,a;
a=(char_code<<3) | 0x40;
      for (i=0; i<8; i++)
      {
      RS_CMD();
      delay_ms(1);
      DR=a;
```

```
        a++;
        Toggle_E(0);
        lcd_putchar(0,*pc++);
        }
RS_CMD();
delay_ms(1);
DR=0b10000000;
Toggle_E(0);
}


/************************************************************************
**********************
************************
************************
************************          I2C EEPROM FUNCTIONS
************************
************************
************************
************************************************************************
*********************/


// Reads data form eeprom addr and returns it
unsigned char eeprom_read(unsigned char addressh,unsigned char addressl)
{
unsigned char data;
i2c_start();
i2c_write(EEPROM_ADDR);
i2c_write(addressh);
i2c_write(addressl);
i2c_start();
i2c_write(EEPROM_ADDR | 1);
data=i2c_read(0);
i2c_stop();
return data;
}

// Writes data to eeprom addr
void eeprom_write(unsigned char addressh,unsigned char addressl, unsigned char data)
{
i2c_start();
i2c_write(EEPROM_ADDR);
i2c_write(addressh);
i2c_write(addressl);
i2c_write(data);
```

```
i2c_stop();
delay_ms(10);
}

// Writes a String to eeprom, useful for inputing text to memory
void eeprom_write_string(int addrh,int addrl, char flash *str)
{
 int n=0;
        while(n<strlenf(str))         // Transfer individual atoms of the string
                {
                eeprom_write(addrh,addrl+n,str[n]);
                n++;
                }
}

// Stores a drink with the givne ingrediants and names to EEPROM
void store_drink(int drink_index, char flash *str1,char flash *str2, int i1, int i2, int i3, int
i4, int i5, int i6)
{
eeprom_write_string(drink_index,DRINK_NAME,str1);
eeprom_write_string(drink_index,SPOKEN_NAME,str2);
eeprom_write(drink_index,VODKA_AMT,i1);
eeprom_write(drink_index,BLUE_AMT,i2);
eeprom_write(drink_index,MIDORI_AMT,i3);
eeprom_write(drink_index,PEACH_AMT,i4);
eeprom_write(drink_index,CRAN_AMT,i5);
eeprom_write(drink_index,OJ_AMT,i6);
}

//Programs all fo the drinks and their ingrediants into EEPROM
void Program_EEPROM()
{
int index=0;

store_drink(index,"Cranberry Newt   ","Cranbearry Newt  ",3,0,0,0,3,3);
index++;
store_drink(index,"Fuzzy Peach      ","Fuzzy Peach      ",3,0,0,3,0,3);
index++;
store_drink(index,"Blue Peach       ","Blue Peach       ",0,2,0,2,0,0);
index++;
store_drink(index,"Jolly Rancher    ","Jolly Rancher    ",3,0,3,0,3,0);
index++;
store_drink(index,"Straight Vodka   ","Straight Vodka   ",1,0,0,0,0,0);
index++;
store_drink(index,"Screwdriver      ","Screw Driver     ",2,0,0,0,0,2);
index++;
```

```
store_drink(index,"Simpson Solution ","Simpson Solution ",3,0,0,3,0,3);
index++;
store_drink(index,"Tootsy Pop       ","Tootsy Pop       ",0,0,3,3,0,3);
index++;
store_drink(index,"Woo Woo          ","Woo Woo          ",3,0,0,3,3,0);
index++;
store_drink(index,"Airhead          ","Airhead          ",0,0,0,2,2,0);
index++;
store_drink(index,"Bad Habit        ","Bad Habit        ",2,0,0,2,0,0);
index++;
store_drink(index,"Fuzzy Navel      ","Fuzzy Navel      ",0,0,0,2,0,2);
index++;
store_drink(index,"Green Russian    ","Green Russian    ",2,0,2,0,0,0);
index++;
store_drink(index,"Skinny Dipper    ","Skinny Dipper    ",0,0,2,0,2,0);
index++;
store_drink(index,"Purple Rain Shot ","Purple Rain Shot ",3,3,0,0,3,0);
index++;
store_drink(index,"Shamrocks        ","Shamrawcs        ",3,3,0,0,0,3);
index++;
store_drink(index,"Green Widow      ","Green Widow      ",0,2,0,0,0,2);
index++;
store_drink(index,"Hairy Sack       ","Hairy Sack       ",0,2,2,0,0,0);
index++;
store_drink(index,"Purple Nurple    ","Purple Nurple    ",0,2,0,0,2,0);
index++;
store_drink(index,"Water            ","Water            ",1,0,0,0,0,0);
}



/*************************************************************************
**********************
************************
************************
************************          SD20 Servo Functions
************************
************************
************************
*************************************************************************
*********************/

// Initializes SD20
void servo_init()
```

```
{
i2c_start();
i2c_write(SERVO_ADDR);
i2c_write(21);
i2c_write(43);
i2c_stop();

i2c_start();
i2c_write(SERVO_ADDR);
i2c_write(22);
i2c_write(0x03);
i2c_write(0x34);
i2c_stop();
}

// Moves servo x to given position
 void move_servo(int num, int pos)
 {

i2c_start();
i2c_write(SERVO_ADDR);
i2c_write(num);
i2c_write(pos);
i2c_stop();
 }

// Sets the speed of the selected motor
void motor(int num, int pos)
{
if(pos==0 || num==1)
        move_servo(num,pos);
else
        move_servo(num,256-pos);

}



/*************************************************************************
**********************
************************
************************
************************          US Ranger Functions
************************
```

```
************************
************************
*************************************************************************
********************/


void ping()
{
static int Data_Pending=0;
int Busy;

i2c_start();
i2c_write(SONAR_ADDR);
i2c_write(0x00);
i2c_start();
i2c_write(SONAR_ADDR | 1);
Busy=i2c_read(0);
i2c_stop();

if(Busy!=255 && Data_Pending==0)
{
i2c_start();
i2c_write(SONAR_ADDR);
i2c_write(0);
i2c_write(0x51);
i2c_stop();
Data_Pending=1;
}

else if(Busy!=255 && Data_Pending==1)
{
i2c_start();
i2c_write(SONAR_ADDR);
i2c_write(0x03);
i2c_start();
i2c_write(SONAR_ADDR | 1);
Ping_Dist=i2c_read(0);
i2c_stop();
Data_Pending=0;
}
}

/*************************************************************************
**********************
************************
************************
```

```
************************          TTS Functions
************************
************************
************************
*******************************************************************************
*********************/

// Speaks a string
void speak(char flash *str)
{
int n=0;

if(strlenf(str)<82)
        {
        i2c_start();
        i2c_write(TTS_ADDR);
        i2c_write(TTS_CMD_REGISTER);
        i2c_write(NOP);
        i2c_write(TTS_Volume);
        i2c_write(TTS_Speed);
        i2c_write(TTS_Pitch);
        i2c_stop();

        i2c_start();
        i2c_write(TTS_ADDR);
        i2c_write(TTS_CMD_REGISTER);
        i2c_write(NOP);
        while(n<strlenf(str))
                {
                i2c_write(str[n]);
                n++;
                }
        i2c_write(TTS_NULL);
        i2c_stop();

        i2c_start();
        i2c_write(TTS_ADDR);
        i2c_write(TTS_CMD_REGISTER);
        i2c_write(SPKBUF);
        i2c_stop();
        }
else
        {
        speak("Error, String too long");
        }
}
```

```
// Speaks a phrase about a givne selection(index)
void speak_selection(int selection)
{
int n=0;
char charholder[20];

while (n<20)
{
charholder[n]=eeprom_read(selection,n+SPOKEN_NAME);
n++;
}

 n=0;
          i2c_start();
          i2c_write(TTS_ADDR);
          i2c_write(TTS_CMD_REGISTER);
          i2c_write(NOP);
          i2c_write(TTS_Volume);
          i2c_write(TTS_Speed);
          i2c_write(TTS_Pitch);
          i2c_stop();

          i2c_start();
          i2c_write(TTS_ADDR);
          i2c_write(TTS_CMD_REGISTER);
          i2c_write(NOP);
          i2c_write('O');
          i2c_write('n');
          i2c_write('e');
          i2c_write(' ');
while (n<20)
{
i2c_write(charholder[n]);
n++;
}


          i2c_write(' ');
          i2c_write('C');
          i2c_write('o');
          i2c_write('m');
          i2c_write('i');
          i2c_write('n');
          i2c_write('g');
```

```
        i2c_write(' ');
        i2c_write('u');
        i2c_write('p');
        i2c_write('!');


        i2c_write(TTS_NULL);
        i2c_stop();

        i2c_start();
        i2c_write(TTS_ADDR);
        i2c_write(TTS_CMD_REGISTER);
        i2c_write(SPKBUF);
        i2c_stop();


}



/*************************************************************************
**********************
************************
************************
************************          MENU Functions
************************
************************
************************
************************************************************************
*********************/


// Displays the name stores at the given index(high byte value) in memory
void display_name(int index)
{
int screen, cntr=0,charholder;

screen=index%5+2;

lcd_clear(screen);
while (cntr<16)
{
charholder=eeprom_read(index,cntr+DRINK_NAME);
lcd_putchar(screen,charholder);
cntr++;
```

```
}
}


// Flashes the ingrediants of the drink stored at the givne index
void display_ing(int index)
{
int screen, cntr=0,charholder;

screen=index%5+2;

lcd_clear(screen);

charholder=eeprom_read(index,VODKA_AMT);
if(charholder!=0)
{
lcd_putsf(screen,"1/");
lcd_putint(screen,charholder);
lcd_putsf(screen," Vodka");
lcd_gotoxy(screen,0,1);
cntr++;
}

charholder=eeprom_read(index,BLUE_AMT);
if(charholder!=0)
{
lcd_putsf(screen,"1/");
lcd_putint(screen,charholder);
lcd_putsf(screen," Blue Caracao");
lcd_gotoxy(screen,0,1);
cntr++;
}

charholder=eeprom_read(index,MIDORI_AMT);
if(charholder!=0)
{
        if(cntr==2)
        {
        delay_ms(1000);
        lcd_clear(screen);
        cntr=0;
        }
lcd_putsf(screen,"1/");
lcd_putint(screen,charholder);
lcd_putsf(screen," Midori");
lcd_gotoxy(screen,0,1);
```

```
cntr++;
}

charholder=eeprom_read(index,PEACH_AMT);
if(charholder!=0)
{
        if(cntr==2)
        {
        delay_ms(1000);
        lcd_clear(screen);
        cntr=0;
        }
lcd_putsf(screen,"1/");
lcd_putint(screen,charholder);
lcd_putsf(screen," Peach Schnap");
lcd_gotoxy(screen,0,1);
cntr++;
}


charholder=eeprom_read(index,CRAN_AMT);
if(charholder!=0)
{
        if(cntr==2)
        {
        delay_ms(1000);
        lcd_clear(screen);
        cntr=0;
        }
lcd_putsf(screen,"1/");
lcd_putint(screen,charholder);
lcd_putsf(screen," Cranberry");
lcd_gotoxy(screen,0,1);
cntr++;
}


charholder=eeprom_read(index,OJ_AMT);
if(charholder!=0)
{
        if(cntr==2)
        {
        delay_ms(1000);
        lcd_clear(screen);
        cntr=0;
        }
```

```
lcd_putsf(screen,"1/");
lcd_putint(screen,charholder);
lcd_putsf(screen," Orange Juice");
lcd_gotoxy(screen,0,1);
cntr++;
}

delay_ms(1000);

display_name(index);
}



/*****************************************************************************
**********************
*************************
*************************
*************************          Switch Functions
*************************
*************************
*************************
*****************************************************************************
*********************/


// Reads the switches ns tores their values into global vars
void read_switches()
{
i2c_start();
i2c_write(SWITCH_ADDR_LOW | 1);
Switch_Status_low=i2c_read(0);
i2c_stop();
i2c_start();
i2c_write(SWITCH_ADDR_HIGH | 1);
Switch_Status_high=i2c_read(0);
i2c_stop();
}

// Presents a dynamic display to the user and takes their selection
void get_selection()
{
lcd_clear(0);
speak("Please Select a shot");
lcd_putsf(1,"Select a shot:");
```

```
display_name(drindex++);
display_name(drindex++);
display_name(drindex++);
display_name(drindex++);
display_name(drindex++);

while(selection==999)
{

read_switches();

        if((Switch_Status_low & 0b00000001) ==1)
        {
        if(drindex==20) {drindex=0; mod=-5;}
        display_name(drindex++);
        display_name(drindex++);
        display_name(drindex++);
        display_name(drindex++);
        display_name(drindex++);
        mod=mod+5;
        }

        if((Switch_Status_low & 0b00000010) == 2)
        display_ing(0+mod);

        if((Switch_Status_low & 0b00000100) == 4)
        display_ing(1+mod);

        if((Switch_Status_low & 0b00001000) == 8)
        display_ing(2+mod);

        if((Switch_Status_low & 0b00010000) == 16)
        display_ing(3+mod);

        if((Switch_Status_low & 0b00100000) == 32)
        display_ing(4+mod);

        if((Switch_Status_high & 0b00000001) ==1)
        selection=0+mod;

        if((Switch_Status_high & 0b00000010) ==2)
        selection=1+mod;

        if((Switch_Status_high & 0b00000100) ==4)
        selection=2+mod;
```

```
        if((Switch_Status_high & 0b00001000) ==8)
        selection=3+mod;

        if((Switch_Status_high & 0b00010000) ==16)
        selection=4+mod;

      if(selection!=999)
      {
      lcd_clear(0);
      speak_selection(selection);
      }

}

}



/*************************************************************************
**********************
*************************
*************************
*************************          Alarm Functions
*************************
*************************
*************************
*************************************************************************
*********************/


// Sets off the alarm for 1 second
 void alarm()
 {
lcd_clear(0);
lcd_putsf(0,"    ALARM!!!    ");
i2c_start();
i2c_write(ALARM_ADDR);
i2c_write(0xFF);
i2c_stop();
delay_ms(1000);
i2c_start();
i2c_write(ALARM_ADDR);
i2c_write(0x00);
i2c_stop();
while(1);
}
```

```
// Initialize the alarm to off
void alarm_init()
{
 i2c_start();
i2c_write(ALARM_ADDR);
i2c_write(0x00);
i2c_stop();
}



/*************************************************************************
**********************
************************
************************
************************        Token Validator Functions
************************
************************
************************
**********************************************************************
**********************/


// Collect and validate the token
void token_validate()
{
int testervalue,valid_token=0;

speak("Please insert a token");
lcd_clear(0);
lcd_putsf(1," Please Insert");
lcd_gotoxy(1,0,1);
lcd_putsf(1,"    a Token");


while(valid_token==0){
i2c_start();
i2c_write(TOKEN_A2D_ADDR);
i2c_write(0b10000100);
i2c_start();
i2c_write(TOKEN_A2D_ADDR | 1);
a2d_high=i2c_read(0);
a2d_low=i2c_read(0);
i2c_stop();
```

```
testervalue=a2d_high;

i2c_start();
i2c_write(TOKEN_A2D_ADDR);
i2c_write(0b10010100);
i2c_start();
i2c_write(TOKEN_A2D_ADDR | 1);
a2d_high=i2c_read(0);
a2d_low=i2c_read(0);
i2c_stop();

if(a2d_high<15)
{
lcd_putsf(2,"    FRAUD!    ");
delay_ms(500);
lcd_clear(2);
valid_token=2;
}
else if(testervalue<15)
{
lcd_putsf(2,"    VALID!    ");
speak("Thank you!");
delay_ms(3500);
lcd_clear(2);
valid_token=1;
}

delay_ms(30);
}

if(valid_token==2)
alarm();
}



void read_IR_and_bump()
{

i2c_start();
i2c_write(EOB_ADDR | 1);
Readings=i2c_read(0);
i2c_stop();

IR_Front=Readings & 0b00000001;
```

```
IR_Left=(Readings & 0b00000010)/2;
IR_Right=(Readings & 0b00000100)/4;

Front_Bump=(Readings & 0b01000000)/64;
Top_Bump=(Readings & 0b00100000)/32;

}


void pan()
{
static int Pan_Counter=0, Position=0;

if(Pan_Counter==3)
{
Pan_Counter=0;

          if(Position==0)
          {
          Position++;
          move_servo(3,200);
          }
          else if(Position==1)
          {
       Position++;
          move_servo(3,128);
          }
          else if(Position==2)
          {
       Position++;
          move_servo(3,56);
          }
          else if(Position==3)
          {
       Position=0;
          move_servo(3,128);
          }
}

else
Pan_Counter++;


}
```

```
void arbitrate_movement()
{

while(Top_Bump==0)
{
ping();
read_IR_and_bump();


pan();
/*
lcd_clear(0);
lcd_putsf(1,"Readings: ");
lcd_putint(1,Readings);
lcd_putsf(2,"Left: ");
lcd_putint(2,IR_Left);
lcd_putsf(3,"Right: ");
lcd_putint(3,IR_Right);
lcd_putsf(4,"Ping Dist:");
lcd_putint(4,Ping_Dist);
lcd_putsf(5,"Top Bump:");
lcd_putint(5,Top_Bump);
lcd_putsf(6,"Frnt Bump:");
lcd_putint(6,Front_Bump);
*/
// See if sonar picks up and obstacle
if(Ping_Dist<135)
{
speak("Please remove obstacle");
move_servo(1,0);
move_servo(2,0);
while(Ping_Dist<142)
ping();
speak("Thank you!");
}



// Don't go if front bump is pressed
if(Front_Bump==1)
{
speak("Please remove obstacle");
move_servo(1,0);
move_servo(2,0);
while(Front_Bump==1)
```

```
read_IR_and_bump();
speak("Thank you!");
}

// 90 deg corner routine
if(IR_Front==0 && IR_Left==0 && IR_Right==0)
{
motor(LEFT_MOTOR,1);
motor(RIGHT_MOTOR,180);
while(IR_Left==0 && IR_Right==0)
read_IR_and_bump();
}


// Straight edge following routines
if(IR_Front==1 && IR_Left==1 && IR_Right==1)
{
motor(LEFT_MOTOR,255);
motor(RIGHT_MOTOR,140);
}
if(IR_Front==1 && IR_Left==0 && IR_Right==0)
{
motor(LEFT_MOTOR,140);
motor(RIGHT_MOTOR,255);
}
if(IR_Front==1 && IR_Left==1 && IR_Right==0)
{
motor(LEFT_MOTOR,255);
motor(RIGHT_MOTOR,220);
}

if(Top_Bump==1)
{
motor(LEFT_MOTOR,255);
motor(RIGHT_MOTOR,70);
delay_ms(1000);

}

delay_ms(50);

}
move_servo(1,0);
move_servo(2,0);
```

```
}



// External Interrupt 0 service routine
interrupt [EXT_INT2] void ext_int2_isr(void)
{

static unsigned char data;// Holds the received scan code
static int index=0;

if (!edge) // Routine entered at falling edge
{
if(bitcount < 11 && bitcount > 2)// Bit 3 to 10 is data. Parity bit,
{ // start and stop bits are ignored.
data = (data >> 1);
if(PIND & 8)
data = data | 0x80;// Store a '1'
}
EICRA = RISING_EDGE;// Set interrupt on rising edge
edge = 1;
} else { // Routine entered at rising edge
EICRA = FALLING_EDGE;// Set interrupt on falling edge
edge = 0;
if(--bitcount == 0)// All bits received
{
lcd_putint(6,data);

Scanned_Code[index]=data;
index=index+1;
if(index==12)
{
index=0;
Code_Ready=1;
}
bitcount = 11;
}
}

}


void init_kb(void)
```

```
{
inpt = kb_buffer;// Initialize buffer
outpt = kb_buffer;
buffcnt = 0;
EICRA = FALLING_EDGE; // INT0 interrupt on falling edge
edge = 0; // 0 = falling edge 1 = rising edge
bitcount = 11;
}



void filter_scanned()
{
int n=0;

while(n<12)
{
        if(Scanned_Code[n]==69)
        Scanned_Code[n]=0;

if(Scanned_Code[n]==0x16)
Scanned_Code[n]=1;

if(Scanned_Code[n]==30)
Scanned_Code[n]=2;

if(Scanned_Code[n]==0x26)
Scanned_Code[n]=3;

if(Scanned_Code[n]==0x25)
Scanned_Code[n]=4;

if(Scanned_Code[n]==0x2e)
Scanned_Code[n]=5;

if(Scanned_Code[n]==0x36)
Scanned_Code[n]=6;

if(Scanned_Code[n]==61)
Scanned_Code[n]=7;

if(Scanned_Code[n]==0x3e)
Scanned_Code[n]=8;

if(Scanned_Code[n]==70)
Scanned_Code[n]=9;
```

```
if(Scanned_Code[n]!=Correct_Code[n])
Valid_ID=0;

n++;
}
}


void check_id()
{
lcd_clear(0);
speak("Please swipe wrist band");
lcd_putsf(1,"  Please Swipe");
lcd_gotoxy(1,0,1);
lcd_putsf(1,"   Wristband");

// Global enable interrupts
#asm("sei")


 while(Code_Ready!=1);
#asm("cli")
filter_scanned();

lcd_putint(2,Scanned_Code[0]);
lcd_putint(2,Scanned_Code[1]);
lcd_putint(2,Scanned_Code[2]);
lcd_putint(2,Scanned_Code[3]);
lcd_putint(2,Scanned_Code[4]);
lcd_putint(2,Scanned_Code[5]);
lcd_putint(2,Scanned_Code[6]);
lcd_putint(2,Scanned_Code[7]);
lcd_putint(2,Scanned_Code[8]);
lcd_putint(2,Scanned_Code[9]);
lcd_putint(2,Scanned_Code[10]);
lcd_putint(2,Scanned_Code[11]);

if(Valid_ID==0)
{
lcd_putsf(2,"    FRAUD!    ");
delay_ms(500);
lcd_clear(2);
alarm();
}
```

```
else
{
lcd_clear(2);
lcd_putsf(2,"     VALID!     ");
speak("Thank you!");
delay_ms(3500);
lcd_clear(2);
}
}

void on_pump(int pumpnum, int dly)
{
int cntr=0;

while(cntr*40<dly)
{
i2c_start();
i2c_write(PUMP_ADDR);
i2c_write(0xFF-pow(2,pumpnum-1));
i2c_stop();
delay_ms(10);          //was 10
i2c_start();
i2c_write(PUMP_ADDR);
i2c_write(0xFF);
i2c_stop();
delay_ms(30);   //was 30
cntr++;
}

}


void pour_shot()
{
int HALF_POUR_TIME=3000, THIRD_POUR_TIME=2000,
ALL_POUR_TIME=3000;
int Amount;
Amount=eeprom_read(selection,VODKA_AMT);
on_pump(1,ALL_POUR_TIME/Amount);

Amount=eeprom_read(selection,BLUE_AMT);
on_pump(2,ALL_POUR_TIME/Amount);
Amount=eeprom_read(selection,MIDORI_AMT);
on_pump(3,ALL_POUR_TIME/Amount);
Amount=eeprom_read(selection,PEACH_AMT);
on_pump(4,ALL_POUR_TIME/Amount);
```

```
Amount=eeprom_read(selection,CRAN_AMT);
on_pump(5,ALL_POUR_TIME/Amount);
Amount=eeprom_read(selection,OJ_AMT);
on_pump(6,ALL_POUR_TIME/Amount);
}



/*****************************************************************************
**********************
*************************       |   |  /\   ----  | |
**************************
*************************       |\  /|  /--\   ||   |\ |
**************************
*************************       | \/ | /    \ ----  | \|
**************************
*****************************************************************************
**********************/

void main(void)
{

PORTC=0x00;
DDRC=0xFF;

// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Low level
// INT1: Off
// INT2: Off
// INT3: Off
// INT4: Off
// INT5: Off
// INT6: Off
// INT7: Off
EICRA=0x00;
EICRB=0x00;
EIMSK=0x04;
EIFR=0x04;


i2c_init();
alarm_init();
lcd_init();
```

```
// Gives time for coprocessors to initialize themselves
delay_ms(500);

// Initializations of I2C, servos, and LCD system

servo_init();

init_kb();

 define_char(char0,0);
 /*

move_servo(TURNTABLE_SERVO,1);
delay_ms(2000);
move_servo(TURNTABLE_SERVO,255);
delay_ms(2000);
move_servo(TURNTABLE_SERVO,1);
*/

// Wait for a switch hit before the robot starts to move


/*while(1){

move_servo(TURNTABLE_SERVO,1);
lcd_clear(1);
lcd_putint(1,1);
delay_ms(1000);
read_switches();

while (Switch_Status_low==0)
read_switches();


move_servo(TURNTABLE_SERVO,255);
lcd_clear(1);
lcd_putint(1,240);
delay_ms(1000);
read_switches();
while (Switch_Status_low==0)
read_switches();


}
*/
```

```
// This is the routine to program the EEPROM, do not uncomment unless you want to
reprog EEPROM
// Program_EEPROM();



read_switches();

while (Switch_Status_low==0)
read_switches();

lcd_backlight(OFF);

lcd_clear(0);
lcd_putsf(1,"    ");
lcd_putchar(1,0);
lcd_putsf(1, " SHOTBOT ");
lcd_putchar(1,0);
lcd_putsf(2," Tap my shoulder");
lcd_gotoxy(2,0,1);
lcd_putsf(2, "  for service!");

arbitrate_movement();

lcd_backlight(ON);
check_id();

token_validate();

get_selection();

delay_ms(6000);
speak("Place a shotglass in the holder please");

move_servo(TURNTABLE_SERVO,1);

lcd_clear(0);
lcd_putsf(1," Place a glass  ");
lcd_gotoxy(1,0,1);
lcd_putsf(1," in the holder");
lcd_putsf(2,"  Hit MORE when");
lcd_gotoxy(2,0,1);
lcd_putsf(2," glass is ready");
```

```
read_switches();

while((Switch_Status_low & 0b00000001) !=1)
read_switches();

lcd_clear(0);
lcd_putsf(1,"Loading Data...");

move_servo(TURNTABLE_SERVO,240);
delay_ms(4000);

lcd_clear(0);
lcd_putsf(1,"Pouring...");
pour_shot();


delay_ms(1000);

lcd_clear(0);
speak("Enjoy!");
lcd_putsf(1,"Enjoy!");
lcd_putsf(2,"Hit MORE when");
lcd_gotoxy(2,0,1);
lcd_putsf(2,"you have shot");

move_servo(TURNTABLE_SERVO,1);

read_switches();

while((Switch_Status_low & 0b00000001) !=1)
read_switches();

lcd_clear(0);
move_servo(TURNTABLE_SERVO,240);
delay_ms(2000);

     lcd_backlight(OFF);
read_switches();

while((Switch_Status_low & 0b00000001) !=1)
read_switches();

// PUMP CONTROL TEST CODE
/*
while(1)
```

```
{
i2c_start();
i2c_write(PUMP_ADDR);
i2c_write(0x00);
i2c_stop();
delay_ms(10);
i2c_start();
i2c_write(PUMP_ADDR);
i2c_write(0xFF);
i2c_stop();
delay_ms(30);
}
*/


/*

while (1)
     {

if(temp==1 && hold==0)
{
move_servo(3,200);
temp++;
//delay_ms(500);
}

else if(temp==2 && hold==0)
{
move_servo(3,128);
temp++;
//delay_ms(500);
}

else if(temp==3 && hold==0){
move_servo(3,56);
temp++;
//delay_ms(500);
}

else if(temp==4 && hold==0){
move_servo(3,128);
temp=1;
//delay_ms(500);
}
 delay_ms(300);
```

```
    ping();


 if(Ping_Dist < 135)
 {
 lcd_clear(1);
 lcd_putsf(1,"Obstacle in way");
 hold=1;
 motor(1,128);
 motor(2,128);
 if(Speech_Holder==0)
{
speak("Please move Obstacle");
Speech_Holder=1;
}

    }

    if(Ping_Dist > 140)
    {

        lcd_clear(1);
        lcd_putsf(1,"Path is Clear");
        hold=0;
        motor(1,255);
motor(2,255);

if(Speech_Holder==1)
{
speak("Thank you");
Speech_Holder=0;
}
    }


    }  */

}
```