Charles P. Hoppe II

Thirst Quencher

EEL5666: Intelligent Machine Design Laboratory

10 April 2009

Report #3

FINAL REPORT

Instructors: Dr. A. Antonio Arroyo

Dr. Eric M. Schwartz

TAs: Mike Pridgen

Thomas Vermeer

Table of Contents **Page**

Abstract

The Thirst Quencher is an autonomous agent that can search for the nearest soda machine and retrieve a beverage for its user. The robot uses an array of sensors to govern a set of behaviors in order to accomplish its task. The platform has two levels and is triangular in shape. The lower level is a base for the motors, batteries and the circuit board while the upper level is a base for the mechanical arms and CMU camera. Other sensors or located throughout the vehicle. One arm is used to insert coins and select a beverage choice and the other arm retrieves the container from the dispensing port. The motors are setup in a differential drive pattern for greater mobility. The sensor suite includes bump sensors, IR detectors, and a CMU camera. The sensors provide data to govern the seven behaviors of the Thirst Quencher (Search, Avoid, Escape, Follow, Align, Select and Retrieve). The robot has been constructed and programmed to successfully avoid obstacles by sampling the sensor suite. The special sensor (CMUcam 2) has been implemented in the robot to search for a given color and drive towards it.

Executive Summary

The Thirst Quencher is a retriever type robot. Its use is to fetch a drink from the nearest soda machine. It completes this task by using an array of sensors to control behaviors that govern its actions. The vehicle is equipped with a CMUcam2, bump switches and IR sensors to receive information from its environment. These sensors are mounted to a tri-level mobile platform. The obstacle avoidance sensors (IR and bump) are mounted to the lowest level of the platform. The first level is also host to the integrated circuit board, motor drivers, wheel assemblies and battery. The sensors used to align the robot with the soda machine are located on the second level, as well as the retrieval arm. Attached to the third level is the selection solenoid, CMUcam to track a light beacon, and coin insertion device.

The Thirst Quencher searches for a beacon of light indicting the position of the soda machine with its CMUcam. As it searches, it avoids obstacles that maybe in its path based on inputs from the IR and bump sensors. Once it reaches the beacon, it aligns with the soda machine using the alignment bump sensors. After it is square with the machine, the retrieval arm extends, dropping the retrieval cup into the port of the soda machine. Once the cup is in place, the coin insertion device is activated and the selection solenoid fires. The retrieval arm is retracted and the vehicle backs away from the machine. The robot then returns to the user so that he or she may drink the beverage.

Introduction

Have you ever been in a lab, working on a difficult task and found yourself in need of liquid refreshment? Why walk down the hall to the soda machine and waste precious time when you could have a robot that would retrieve a soda for you? That is what the Thirst Quencher will do. The Thirst Quencher will search out the nearest soda machine, select your favorite type of beverage, retrieve the soda from the machine and return it to you in your lab without you having to take a step.

The Thirst Quencher will use its onboard array of sensors to autonomously search for the nearest soda machine. It will insert the correct change into the coin slot, select a beverage option with its selector solenoid, retrieve the soda from the port with its retrieval arm and return it to the user with its sturdy and dependable mobile platform. This report lays out the specifications of the platform, actuators, sensors and behaviors of the Thirst Quencher robot. The Thirst Quencher can be preprogrammed to choose your favorite beverage and even pay the correct change for the drink.

The object of the project is to integrate information from a suite of sensors to govern behaviors that in turn accomplish a specific task determined by the programmer. The Thirst Quencher uses bump and IR sensors combined with a CMU camera to control seven behaviors. These seven behaviors (Search, Avoid, Escape, Follow, Align, Select and Retrieve) allow the robot to retrieve a beverage from the nearest soda machine.

Integrated System

The Thirst Quencher is a beverage retrieving robot. The requirements of a retrieving robot are a stable, mobile platform and a manipulator of some sort. The autonomous vehicle must also be able to navigate through its environment while avoiding obstacles which would prevent it from completing its mission. This robot has a triangular platform with differential steering. There are two mechanical manipulators, one to insert money into the soda machine (selection arm) and a second arm to retrieve the drink container from the machine (retrieval arm). The Thirst Quencher is fitted with a variety of sensors to communicate with its environment. These sensors, as well as the mechanical linkages, are controlled by an ATmega128 integrated circuit board.

The vehicle uses infrared detectors and bump sensors for collision avoidance and obstacle detection. The robot senses potential obstacles with the IR detectors and corrects its trajectory accordingly, but if this fails then the bump sensors will signal contact with a foreign object and the robot will vary its path to avoid the obstacle. Bump sensors are also used to provide environmental feedback to the Thirst Quencher's computer. Two bump sensors are placed on an extension of the chassis and are activated when the robot is in the correct operating position in respect with the soda machine. When the switches are depressed in tandem, the actions of the selection arm begin.

The CMU camera is used to find a blue LED indicator light on the soda machine. When the camera finds the light, the robot is then instructed to follow a path towards the machine. After the drink container is obtained, the robot reverses course and returns to its original position, thereby retrieving a drink for its user. The next page depicts an illustration of the behavioral hierarchy of the Thirst Quencher.

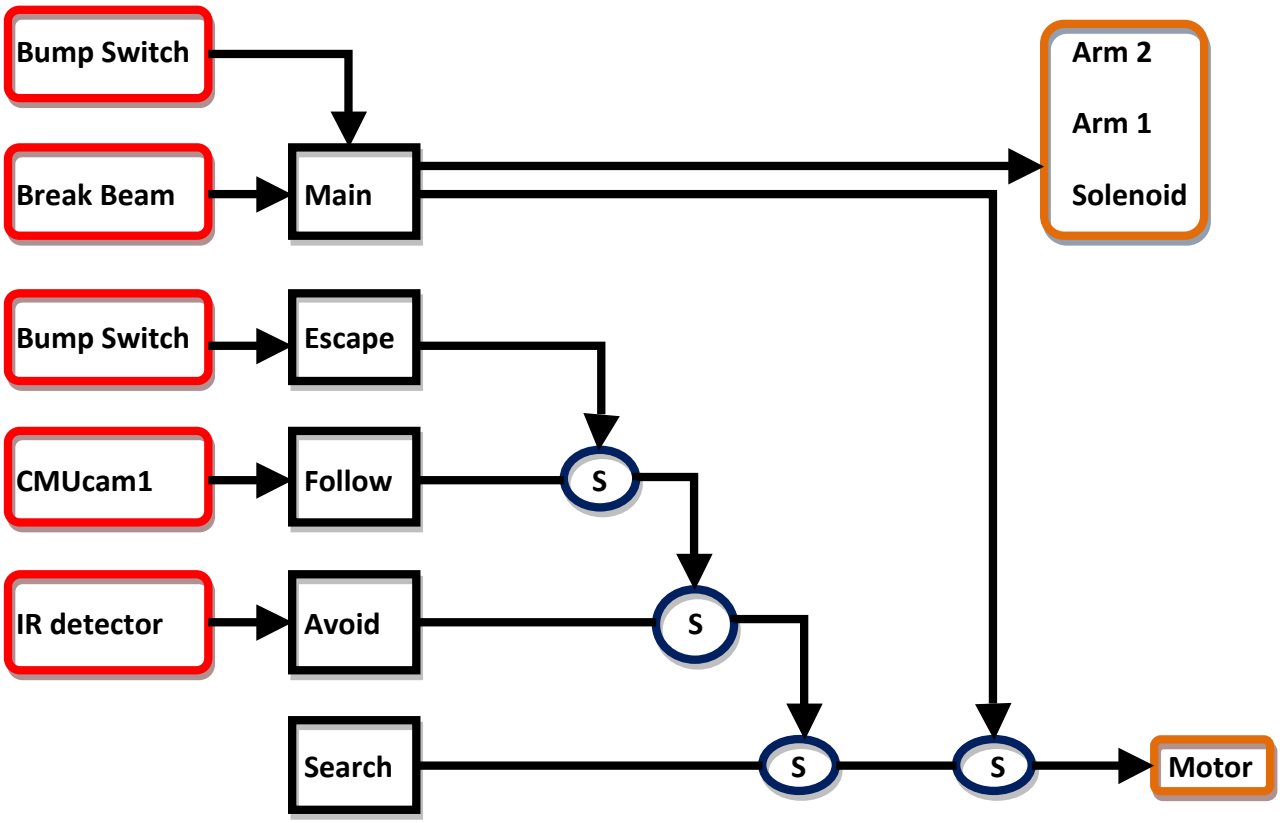


Figure 1. Behavior control program for the Thirst Quencher.

Figure 1 illustrates the behaviors of the Thirst Quencher and the priority of each behavior. The figure also defines which sensors each behavior samples. The lowest-level behavior is Search, which causes the vehicle to roam its environment while searching for soda machine. Avoid is a higher-level program that samples the IR detectors to sense upcoming obstacles. When an obstacle is detected, the Avoid behavior suppresses the Search behavior and sends commands to the motors to maneuver out of the way. The Follow behavior is initialized when the CMUcam detects the LEDs mounted on the soda machine. It suppresses the Search and Avoid behaviors and directs the robot to the light source. If a bump switch is triggered, the Escape behavior causes the Thirst Quencher to back up and change direction. Because the Align, Select and Retrieve behaviors occur one after the other, they can be grouped as the Main super-behavior. This behavior samples bump switches and the break beam to run through the process of retrieving the beverage container from the machine.

Mobile Platform

The Thirst Quencher's platform is triangular in shape with two levels. Attached to the first, lower, level is the vehicle's geared motors with wheels, batteries and circuit board. The drive wheels are set up for differential steering so they are located on the left and right corners of the platform. The caster wheel is attached to the rear corner to provide stability. Also attached to the lower level are the object detection and obstacle avoidance sensors. These are located on the lower level to increase the efficiency of detecting low-lying obstacles and to prevent them from coming in contact with the moving mechanical arms.

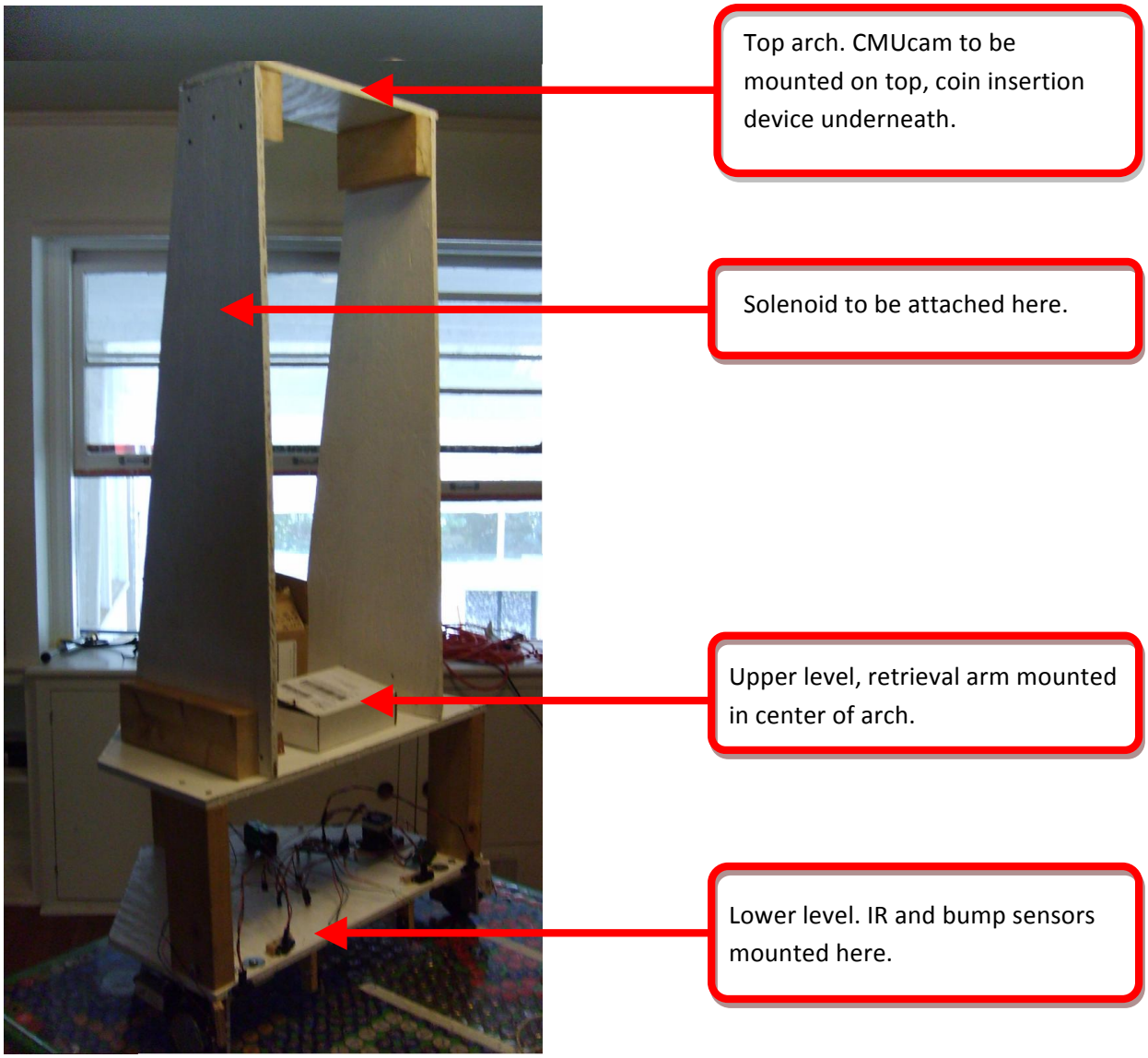


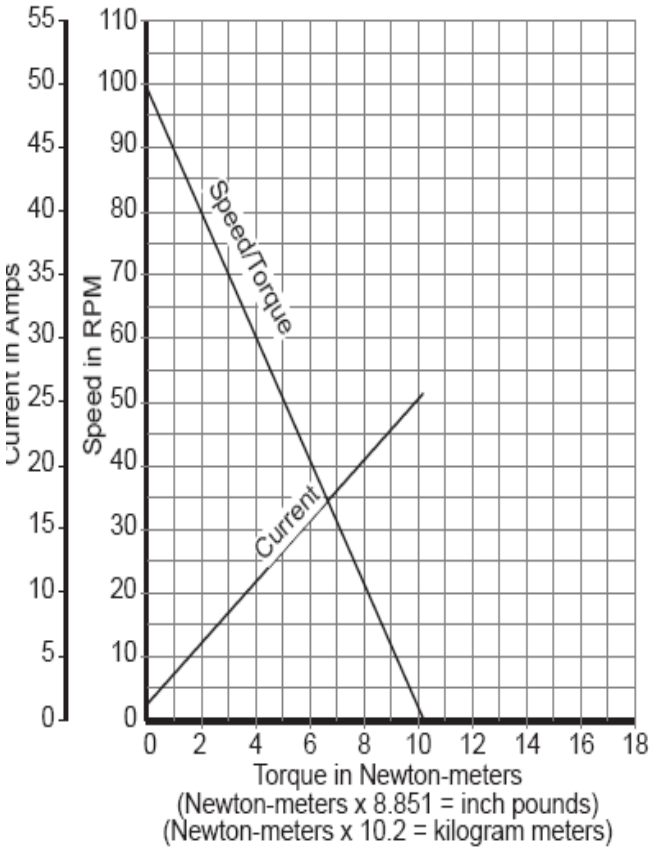
Figure 2. Picture of Thirst Quencher, highlighting the arch for selection arm.

The second, higher level serves as a base for the two mechanical arms. The selection arm is an arch over the platform that allows access to the coin slot and selection buttons on the soda machine. Bump sensors on a projection from the chassis on the second level help align the robot to the correct operating position. The retrieval arm is mounted in the center of the platform in order to index with the dispensing port on the soda machine. The CMU camera is mounted at the top of the arch in order to have the greatest range of vision and easily search for the blue LED indicator lights on the soda machine. The coin insertion device is located at the rear of the arch, behind the CMUcam. The coin slide connects the coin insertion device with the funnel mounted on the coin slot of the soda machine. The selection solenoid is mounted on the side of this arch.

I have learned a great deal from construction of the platform. I learned that if you do not have a good plan for the platform before you begin fabrication, the platform will be a failure. I used scrap wood from around my house to construct my first platform. The size of the robot was determined more by the size of the available wood than the dimensions of the soda machine. Therefore my first attempt was woefully small at the base and severely top heavy. I also learned that machined hubs are more reliable than anything constructed from JB Weld, no matter what the package says. Unfortunately, machined hubs take longer to manufacture, causing delays in getting the robot up and running.

Actuation

The Thirst Quencher's drive train is set up for differential steering. In this configuration, two drive motors are located near the center axis of the vehicle and a third caster wheel is used to provide stability. This design allows the robot to turn on axis and provides great mobility for the platform. A differential drive is used on the Thirst Quencher because accurate steering is needed to align the robot with the soda machine. The right-angle power window motors used to drive the vehicle are 210 series motors from AM Equipment (the 210-1009 right hand and 210-1010 left hand motors). Both provide 92.1 inch pounds of stall torque, but the design calls for a total vehicle weight no more than 30 pounds.



Counter-Clockwise Motor Shaft Rotation		
Data Point	Data Type	Value Range
No Load	Current (A)	3.5 - 2.9
	Speed (rpm)	108.0 - 88.5
Stall Load	Torque (Nm)	11.2 - 9.2
	Current (A)	26.9 - 22.0
Peak Power	Power (W)	29.1 - 23.8
	Torque (Nm)	6.1 - 4.9
Nominal (Peak Efficiency)	Power (W)	21.2 nominal
	Speed (rpm)	67.4 nominal
	Current (A)	8.8 nominal
	Torque (Nm)	3.1 nominal
Clockwise Motor Shaft Rotation		
Data Point	Data Type	Value Range
No Load	Current (A)	3.8 - 3.0
	Speed (rpm)	106.2 - 86.9
Stall Load	Torque (Nm)	11.0 - 9.0
	Current (A)	28.0 - 22.9
Peak Power	Power (W)	30.9 - 25.3
	Torque (Nm)	5.6 - 4.6
Nominal (Peak Efficiency)	Power (W)	24.2 nominal
	Speed (rpm)	70.1 nominal
	Current (A)	9.2 nominal
	Torque (Nm)	3.4 nominal

Figure 3. Speed/Torque Curve for 210 series motors.

Using the information from Figure 3, each motor provides 35.4 inch pounds of torque at a motor speed of 60 rpm. Therefore the Thirst Quencher can move at a forward speed of around 12 inches per second using 4 inch diameter wheels. The robot could move much faster by decreasing the vehicle weight due to the sharp decline of the speed/torque graph. The motors are driven by a Victor 884 from IFI Robotics. Each speed controller supplies a continuous 40A maximum current. The drivers are optically isolated from the circuit board by PC942 photo-couplers from Sharp. There is a schematic drawing of the 210 series motors in the attached Appendix for further review.

There are two mechanical arms on the Thirst Quencher, the selection arm and the retrieval arm. The selection arm is not articulated but serves as a mounting platform for the coin dispenser and the selector solenoid. Before the robot is sent out to retrieve a soda, the coin reservoir is preloaded with the correct amount of change and the selector solenoid is placed in the desired position based on your local soda machine. If the user knows that Mountain Dew is the third selection on the machine and costs $1.00, the selector solenoid would be moved to the third position and the coin reservoir would be stocked with four quarters. When the robot is correctly aligned with the soda machine, the coin reservoir will deposit the change into the soda's coin slot and the selector solenoid will press the selection button on the panel. The coin insertion device consists of a hacked servo that is fitted with a small arm. Located above the arm is the coin reservoir, which empties into a funnel-shaped chute. The servo arm spins, knocking a coin into the chute with each revolution. The coin then travels down the chute into the soda machines coin slot.

The second mechanical arm is articulated and will retrieve the soda container from the dispensing port. When the vehicle is correctly aligned with the soda machine, the arm will deploy to index with the dispensing port. The motor on the arm turns a threaded screw, which in turn causes the arm to move forward. There is a cup attached to the end of the arm by wires. The arm pushes the cup into the retrieval port of the soda machine. The beverage container is then "caught" by this cup, which also serves as the transport container when the Thirst Quencher returns to its user. When the container is caught, it activates a break beam which signals the robot to withdraw the retrieval arm. The arm pulls the cup out of the port by the cables when it retreats. After the arm is withdrawn, the robot back away from the soda machine and returns to the user.

Sensors

The Thirst Quencher uses sensors to communicate with its environment. The different sensors attached to the robot are used to govern the behaviors of the vehicle. Bump sensors control the most behaviors on the Thirst Quencher. Bump sensors located on the lower level of the platform are used to detect obstacles and escape from danger. A bump sensor is activated when the robot runs into a foreign object. This causes the robot to reverse direction and choose a new course to avoid the object. A series of bump sensors is used to determine whether the vehicle is correctly aligned with the soda machine. Two sensors are located on a projection of the vehicle chassis on the upper level of the platform. If one sensor is activated, the robot turns in the direction of the other sensor until both sensors are activated. When both sensors are activated, the robot is in the correct position to select a beverage and retrieve a drink from the machine. After the robot is aligned, the coin reservoir on the selector arm begins inserting coins into the machine, followed by the firing of the selector solenoid to choose a drink preference. Figure 4 illustrates the bump switch schematic.

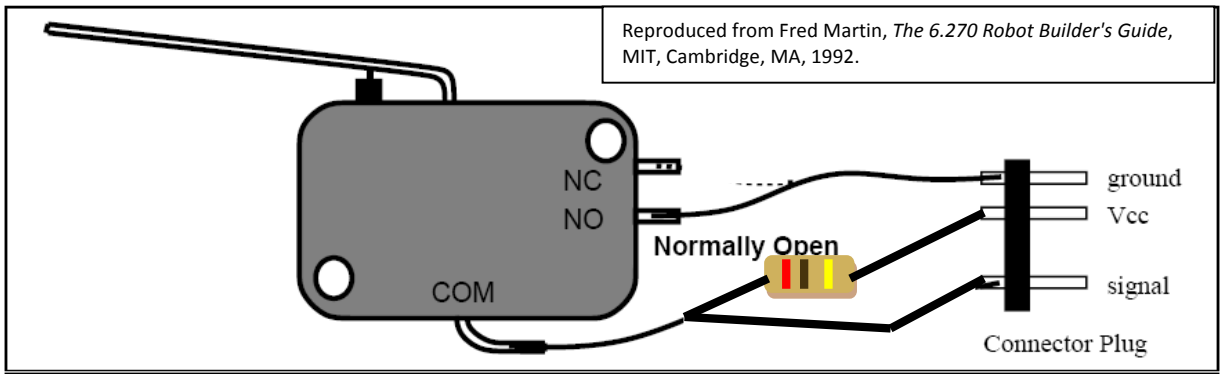


Figure 4. Circuit schematic for bump switches

Infrared detectors are located around the lower level of the Thirst Quencher's platform. The IR detectors are used to avoid collisions with any obstacles in the path of the robot. If the sensor detects the presence of an obstacle, it forces the vehicle to change its heading and move around the object. The IR sensors detect the distance from an object by measuring the amount of reflected light. The robot is programmed to change direction when the amount of reflected light crosses a predetermined threshold value. The circuit schematic for the IR detectors is shown below in Figure 5.

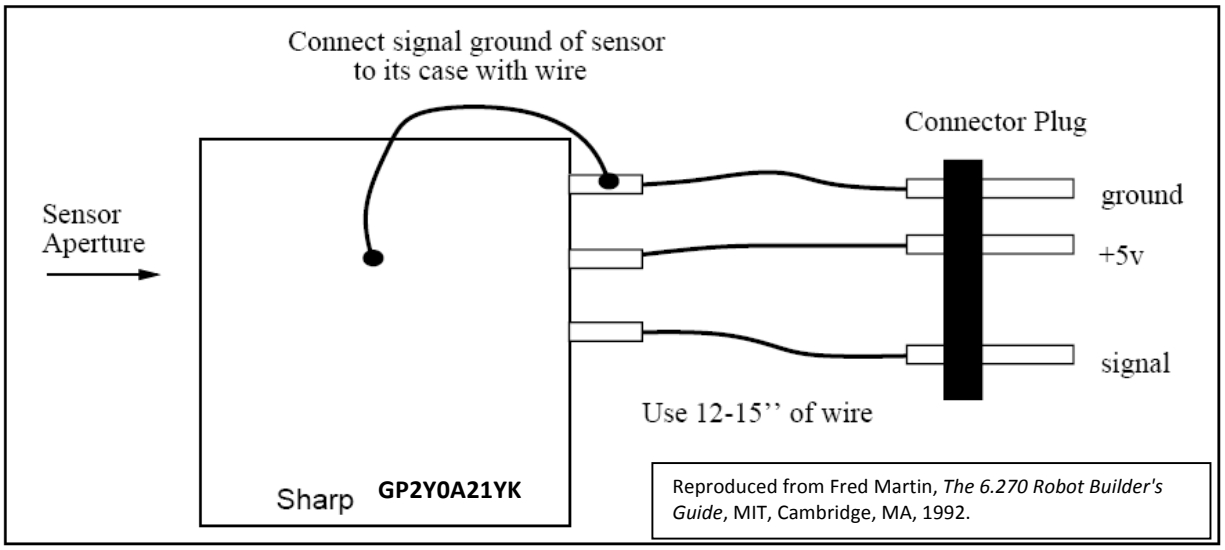


Figure 5. Schematic of Sharp GP2Y0A21YK and connection.

The sensors used on the Thirst Quencher are Sharp GP2Y0A21YKs which have a sensing band of 10 to 80 cm. The analog voltage of the sensor increases as the reflected object moves closer to the IR detector. The voltage is a high of just over 3V at a distance of 10 cm (~4 in) and a low of just under 0.5V at 80 cm (~32 in); this voltage is then converted to a number between 0 and 255 by the analog to digital converter (ADC). The robot is currently programmed with a threshold of 30, which corresponds to 40 cm away (~16 in). This value was chosen because that is depth of the robot; if an object is sensed at that threshold, the vehicle can turn around completely (180 degrees) without hitting the obstacle. Figure 6 presents the results of the IR Sensor Calibration Test, where the digital reading was recorded for each sensor at preselected distances.

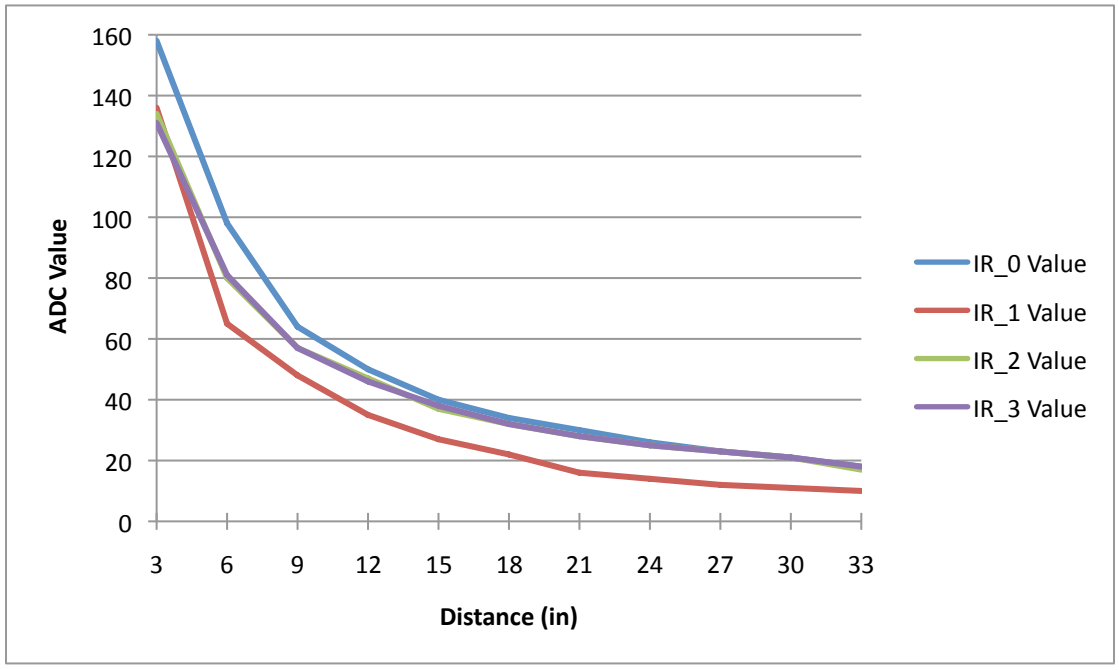


Figure 6. Graph displaying ADC values for each sensor based on distance from object.

The CMU camera is used to search for a bright blue LED indicator light on the soda machine. This camera will be attached to an oscillating servo motor to sweep its surroundings for the LED indicator lights. After the lights are located, the camera's ability to track colors and find the middle mass of the color blob will be used to find the position of the blob. The robot will then drive so that the color blob stays in the center of the camera's frame. This will also help align the robot for docking with the soda machine.

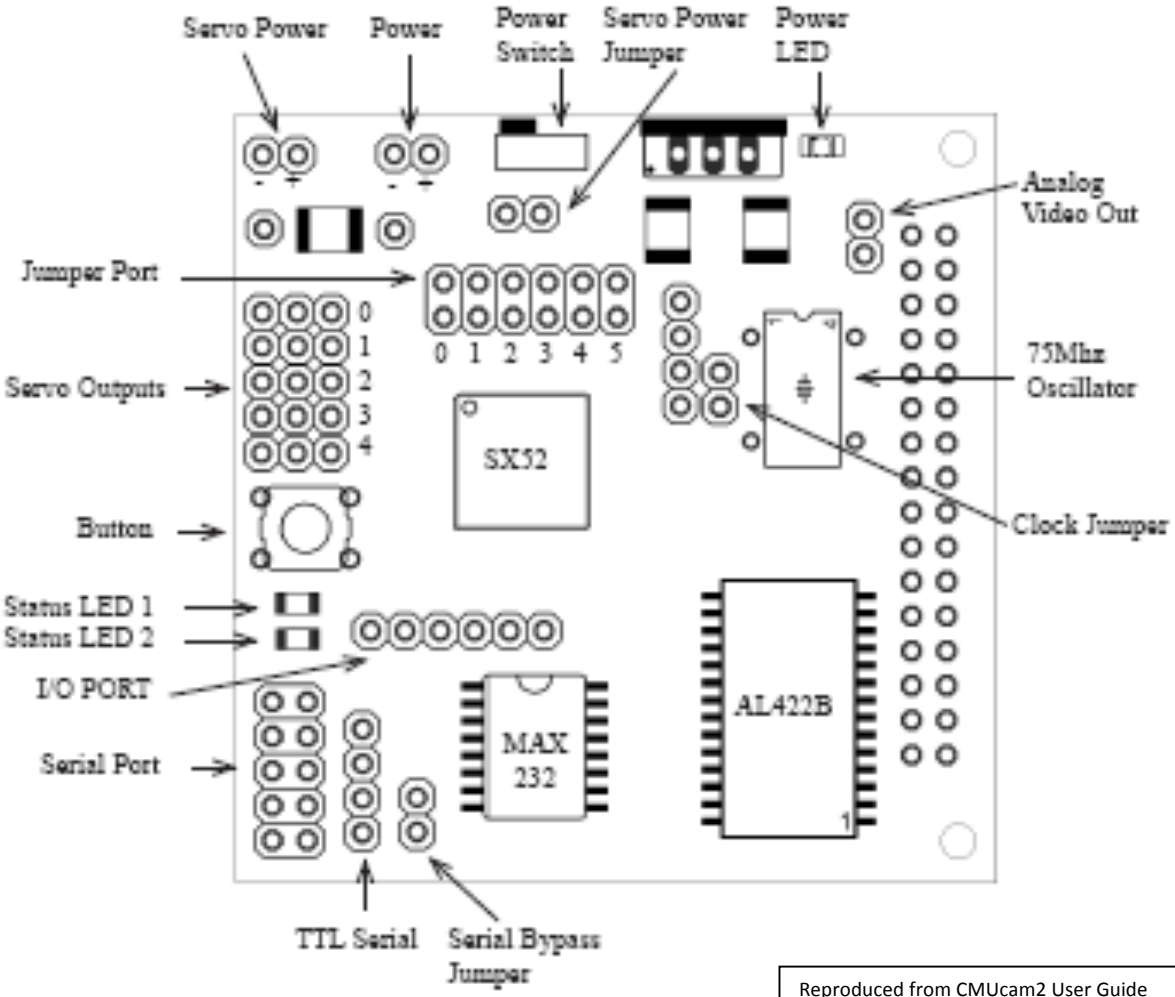


Figure 7. CMUcam2 Board Layout

Behaviors

The Thirst Quencher is initially dormant until the robot is turned on by flipping a toggle switch. This switch also serves as the shutoff for the vehicle. Flipping the toggle initializes the most basic behavior. This is the Search behavior, where the vehicle roams the hallway while searching for the LEDs on the soda machine. While searching for the lights with the CMU camera, if the IR detectors sense an obstacle in the path of the robot, the Search behavior will be suppressed by the Avoid behavior. The Avoid behavior is characterized by adjusting the path of the vehicle to avoid any obstacles. Once the robot successfully navigates around the obstacle, the robot reverts back to the Search behavior.

Once the CMU camera detects the LEDs, the Follow behavior is activated. The robot will follow a straight path to the lights located on the soda machine. The Escape behavior takes precedence when the bump switches are activated. The vehicle will reverse direction and begin on a new path in order to drive around the foreign object. Once around the object the robot returns to the base Search behavior. When the vehicle reaches the soda machine, the bump sensors on the front projection while initialize the Align behavior where the Avoid and Escape behaviors are suppressed and the robots maneuvers into position to retrieve a soda from the soda machine.

When both bump sensors are activated, the Align behavior is suppressed by the Select behavior. The retrieval arm extends into the dispensing port and after a given amount of time, the coin reservoir inserts the change into the coin slot and the selector solenoid depresses the desired button. The Retrieve behavior is activated 10 seconds after the solenoid is fired and the retrieval arm is retracted and the robot backs up from the soda machine. The Align, Select and Retrieve behaviors are combined into one super-behavior as illustrated in Figure 1, located in the Integrated System subsection. Once the vehicle backs away from the soda machine, the robot reverts back to the Search behavior. The CMU camera then searches for another colored LED to indicate where to return the beverage. When the LED is discovered, the Follow behavior is reactivated and the robot returns to its user.

Experimental Layout and Results

Table 1. IR Sensor Calibration Test data:

Distance (in)	IR_0 Value	IR_1 Value	IR_2 Value	IR_3 Value	Distance (cm)
3	158	136	134	131	7.62
6	98	65	80	81	15.24
9	64	48	57	57	22.86
12	50	35	47	46	30.48
15	40	27	37	38	38.1
18	34	22	32	32	45.72
21	30	16	28	28	53.34
24	26	14	25	25	60.96
27	23	12	23	23	68.58
30	21	11	21	21	76.2
33	18	10	17	18	83.82

Conclusion

The Thirst Quencher's platform has been constructed. Attached to the platform are the motors, drivers, IR and bump sensors, as well as the additional 12V battery and circuit board. The CMU camera and the extra bump sensors have been installed as well as the selector and retrieval arms. The robot has performed each behavior separately and the behaviors have been integrated successfully. The robot will display its system of behaviors during demo day.

I believe that my robot is mechanically ambitious but therein lies the challenge and that is why I took the course. I wanted to test my mechanical ability while learning how to design and build an autonomous agent. My electrical design experience can be improved but I am confident that at the end of the semester I will have a working knowledge of autonomous vehicles and how to design more to fit my needs.

If I were to start this project over, I would get approval of my design within the first week or two of the semester so I could get to work right away on designing the platform and ordering parts for the robot. I would also construct a table of necessary parts for each stage of completion so that I would not be waiting weeks to get a part necessary to take the next step. I might also have decided to make a smaller robot that completed fewer tasks. I believe an autonomous vehicle with four integrated tasks is too complicated for a first time robot builder. However, this experience has made me want to build more robots in the future.

Documentation

Fred Martin, *The 6.270 Robot Builder's Guide*, MIT, Cambridge, MA, 1992.

Joseph Jones, Bruce Seiger & Anita Flynn, *Mobile Robots: Inspiration to Implementation*, 2nd edition, A.K. Peters Publishers, Natick, MA, 1998.

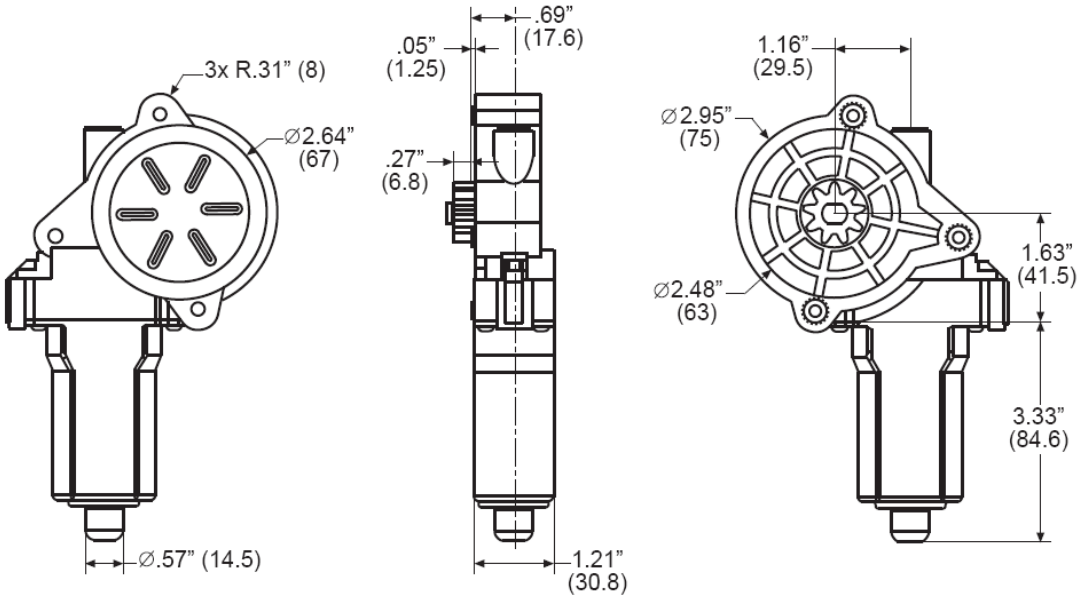
Data Sheets:

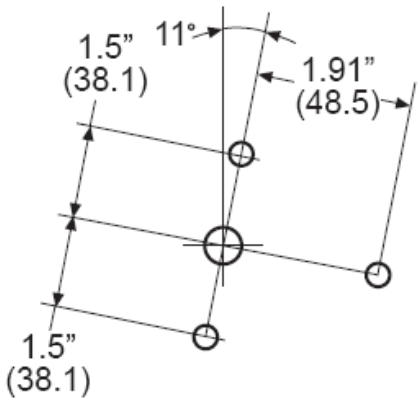
Atmel ATmega128
SeattleRobotics.com CMUcam2 User Guide
Omron G5SB Relay
Xiamen Ocular GDM1602K LCD
Sharp IR GP2Y0A21YK
IFI Victor 884 Speed Controllers

Appendices

I. Spec Sheets

210-1010
- 10Nm stall torque actuator motor, LH
- 12V reversible
- Water resistant
- Weighs 1.3 pounds





Mounting pattern

II. Sample Code

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <string.h>
//#include <avr/pgmspace.h>
#include "sleep.h"
#include "LCD.h"
#include "ADC.h"
#include "PVR_Servos.h"

#define FOSC 16000000// Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1

typedef unsigned int u8;

volatile u8 temp;
volatile u8 i = 0;
volatile u8 cmudat[10];
volatile int tracking_flag = 0;
volatile int align = 0;



//************************************************************
//
//                                                  Initialize USART
//
//************************************************************
void USART_Init(unsigned int ubrr) {

      //
```

```
        //                      !!!  IMPORTANT  !!!
        //
        // **********  REMEMBER TO SET THE JUMPERS FOR BAUD RATE ON THE CMU CAMERA BOARD !! *******



        // Set baud rate registers
        UBRR0L = (unsigned char)ubrr;
        UBRR0H = (unsigned char)(ubrr>>8);

        //Set data frame format: asynchronous mode,no parity, 1 stop bit, 8 bit size
        UCSR0C = (0<<UMSEL0)|(0<<UPM1)|(0<<UPM0)|
                                                (0<<USBS)|(0<<UCSZ2)|(1<<UCSZ1)|(1<<UCSZ0);

        //Enable Transmitter and Receiver and Interrupt on receive complete
        UCSR0B = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE);

        //enable global interrupts
        //set_sleep_mode(SLEEP_MODE_IDLE);
        sei();
}//end



//************************************************************
//
//                                              USART Interrupt Handler
//
//************************************************************
// This code was written by Kyle Tripician (a previous student of IMDL )
ISR(USART0_RX_vect) {

        temp = UDR0;
```

```
        //FlashLight(2);



        // 0x3A = ':'
        // 0x20 = space
        // 0x09 = tab
        // anything less than 9 = other indicators like end of text, etc...
        // 0xFF = a byte received having a value of 255

        if(temp != 0x3A) {
                if(temp == 0x20 || i==10) {
                        i=0;
                }
                else if(i==0){
                        if(temp == 0xFF){
                                cmudat[i]=temp;
                                i++;
                        }
                }
                else if(temp !=0x20 && i<10) {
                        cmudat[i]=temp;
                        i++;
                }
        }//if
}//end



//*****************************************************************
//
//                                      Flash LED
//
//*****************************************************************
void FlashLight(int x){
   DDRB = 0b00000001;
```

```
        int i;
        for (i=0;i<x;x--){
                ms_sleep(100);
                PORTB = 0b11111111;
                ms_sleep(100);
                PORTB = 0b00000000;
                ms_sleep(100);
        }
}


//*****************************************************************
//
//                                                    Send a single byte to USART
//
//*****************************************************************
//Send a single byte of data
void uarttransmit(unsigned char data) {
        // Wait for empty transmit buffer
        while (!(UCSR0A & (1<<UDRE0))){}
        // Put data into buffer, sends the data
        UDR0 = data;
}


//*****************************************************************
//
//                                                    Send a string to USART
//
//*****************************************************************
// Send a given CR terminated string
void uartstring(unsigned char * myStringIn) {
        unsigned char *myString = myStringIn;
        unsigned char ch1;
```

```
	unsigned char gotNULL = 0;
	ch1= *myString++;
	while(!gotNULL){
		uarttransmit(ch1);
		ch1 = *myString++;
		if(ch1 == '\r'){
			gotNULL = 1;
			uarttransmit(ch1);
		}
	}
}


//*************************************************************
//
//                                                  Display contents of an T Packet
//
//*************************************************************
// Display an T packet
void displayTpacket() {

	u8 *mmx, mmy, lcx, lcy, rcx, rcy, pix, conf, packetName;

	// Display the packet vlaues
	packetName = cmudat[1];                  // 'S' = 83, 'M' = 77, 'C' = 67, 'T' = 84
	mmx = cmudat[2];
	mmy = cmudat[3];
	lcx = cmudat[4];
	lcy = cmudat[5];
	rcx = cmudat[6];
	rcy = cmudat[7];
	pix = cmudat[8];
	conf = cmudat[9];
	ms_sleep(2000);
```

```
lcdClear();
lcdGoto(0,0);
lcdString("Packet Name");
lcdGoto(1,0);
//if(cmudat[1] == 83) {
//		lcdString("Packet S");
//}
if(cmudat[1] == 77) {
		lcdString("Packet M");
}
else if(cmudat[1] == 67) {
		lcdString("Packet C");
}
else if (cmudat[1] == 84) {
		lcdString("Packet T");
}
else {
		lcdString("Packet ERROR!");
}
ms_sleep(3000);

lcdClear();
lcdGoto(0,0);
lcdString("Middle Mass X");
lcdGoto(1,0);
lcdInt(mmx);
ms_sleep(3000);

lcdClear();
lcdGoto(0,0);
lcdString("Middle Mass Y");
lcdGoto(1,0);
lcdInt(mmy);
```

```
        ms_sleep(3000);


        lcdClear();
        lcdGoto(0,0);
        lcdString("Pixels");
        lcdGoto(1,0);
        lcdInt(pix);
        ms_sleep(3000);


        lcdClear();
        lcdGoto(0,0);
        lcdString("Confidence");
        lcdGoto(1,0);
        lcdInt(conf);
        ms_sleep(3000);
}


/*********************************************************
*
*                                   Initialize Modules: Sleep, LCD, USART, ADC, SERVOS
*
*********************************************************/
void init_modules() {


        // Initialize modules: SLEEP, LCD, USART, ADC, SERVOS
        initSleep();
        //FlashLight(5);
        lcdInit();
        initADC();
        initServo();
        lcdString("Starting System");
        ms_sleep(2000);
        lcdClear();
        lcdString("Init USART");
```

```
	USART_Init(MYUBRR);
	ms_sleep(3000);
}




/*******************************************************
*
*                                                       Initialize Camera
*
*******************************************************/
void init_camera() {

	// Initialize the camera
	// Reset the camera with several RS commands
	for(int j=0; j<2; j++) {
			uartstring("RS \r");
			lcdClear();
			ms_sleep(200);
			lcdGoto(0,0);
			lcdString("Resetting Camera");
			ms_sleep(500);
	}

	// Turn on auto tracking LED
	uartstring("L1 1\r");
	ms_sleep(5000);

	// To apply a fluroscent band filter and auto lighting adjust
	// when working under fluroscent lighting, use the following command:
	uartstring("CR 45 7 18 44 5 255\r");

	// For normal and incandescent lighting, just use the following command:
```

```
//uartstring("CR 18 44\r");


// wait for 10 seconds to adjust to lighting conditions
for(int j=0; j<10; j++) {
        ms_sleep(1000);
}


// Turn off auto lighting adjust
uartstring("CR 18 40 19 32\r");


// Turn off auto tracking LED
uartstring("L1 2\r");


// Indicate to user to prepare the target by flashing light and sending on LCD
FlashLight(5);
lcdClear();
lcdGoto(0,0);
lcdString("Hold target");
ms_sleep(3000);


// Set poll mode; 1 packet
uartstring("PM 1\r");
ms_sleep(100);


// Set raw mode
uartstring("RM 3\r");
ms_sleep(100);


// issue Track Color command for the color "Blue"
//uartstring("TC 16 50 30 70 100 240\r");
//ms_sleep(100);



// display the command being sent
```

```
	lcdClear();
	lcdGoto(0,0);
	lcdString("Sending TW cmnd");
	ms_sleep(1000);

	// issue Track Window command
	uartstring("TW\r");
	ms_sleep(300);

}



/*******************************************************
*
*                                                    Main program
*
*******************************************************/

int main (void)
{
	lcdClear();
	lcdString("Chas Hoppe");
	lcdGoto(1,0);
	lcdString("ThirstQuencher");
	ms_sleep(3000);

	init_modules();

	DDRA = 0b00000000;
	DDRB = 0b11111111;

	init_camera();
```

```
while(1)
{

if (align == 0);
{

        if ((PINA & 0b00001001) == 0)
        {
                lcdClear();
                lcdString("Bump: Sides");
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(-100);
                moveServo2(-100);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(-50);
                moveServo2(50);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
        }

        if ((PINA & 0b00000110) == 0)
        {
                lcdClear();
                lcdString("Bump: Center");
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(-100);
```

```
		moveServo2(-100);
		ms_sleep(2000);
		moveServo1(-15);
		moveServo2(-15);
		ms_sleep(300);
		moveServo1(-50);
		moveServo2(50);
		ms_sleep(2000);
		moveServo1(-15);
		moveServo2(-15);
	}

	if ((PINA & 0b00000001) == 0)
	{
		lcdClear();
		lcdString("Bump: Right");
		moveServo1(-15);
		moveServo2(-15);
		ms_sleep(300);
		moveServo1(-100);
		moveServo2(-100);
		ms_sleep(2000);
		moveServo1(-15);
		moveServo2(-15);
		ms_sleep(300);
		moveServo1(50);
		moveServo2(-50);
		ms_sleep(2000);
		moveServo1(-15);
		moveServo2(-15);
	}

	if ((PINA & 0b00000010) == 0)
	{
```

```
                lcdClear();
                lcdString("Bump: Right Mid");
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(-100);
                moveServo2(-100);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(50);
                moveServo2(-50);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
}

if ((PINA & 0b00000100) == 0)
{
                lcdClear();
                lcdString("Bump: Left Mid");
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(-100);
                moveServo2(-100);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(-50);
                moveServo2(50);
                ms_sleep(2000);
```

```
                moveServo1(-15);
                moveServo2(-15);
}

if ((PINA & 0b00001000) == 0)
{
                lcdClear();
                lcdString("Bump: Left");
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(-100);
                moveServo2(-100);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(-50);
                moveServo2(50);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
}

else
{
                int motor_l;
                int motor_r;
                int data,pixels,normal;

                lcdClear();
                lcdString("Sending cmnd");
                ms_sleep(1000);
                uartstring("TC \r");//77 63 56
```

```
ms_sleep(100);


data=cmudat[2];
pixels=cmudat[8];
normal=data-80;


if (data != 0)
{
        tracking_flag = 1;


        if(pixels<240)
        {
                if(5>normal && normal>-5)
                {
                        lcdClear();
                        lcdString("Center");
                        motor_l=motor_r=25;
                        moveServo1(motor_r);
                        moveServo2(motor_l);
                        ms_sleep(1000);
                        moveServo1(-15);
                        moveServo2(-15);
                }


                else if(normal>5)
                {
                        lcdClear();
                        lcdString("Left");
                        motor_r=(normal)*(25/80);
                        motor_l=-15;
                        moveServo1(motor_r);
                        moveServo2(motor_l);
                        ms_sleep(1000);
                        moveServo1(-15);
```

```
                moveServo2(-15);
            }

            else if (normal<-5)
            {
                lcdClear();
                lcdString("Right");
                motor_l=(-normal)*(25/80);
                motor_r=-15;
                moveServo1(motor_r);
                moveServo2(motor_l);
                ms_sleep(1000);
                moveServo1(-15);
                moveServo2(-15);
            }
        }
        else
        {
            moveServo1(-15);
            moveServo2(-15);
            ms_sleep(300);
            moveServo1(-40);
            moveServo2(25);
            ms_sleep(2000);
            moveServo1(-15);
            moveServo2(-15);

        }
    }

    else
    {
        lcdClear();
        lcdString("No Object!");
```

```
                motor_l=-40;
                motor_r=25;
                moveServo1(motor_r);
                moveServo2(motor_l);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
        }
}

if (tracking_flag == 0)
{
        lcdClear();
        lcdString("Searching...");

        if (adcZero() > 50)
        {
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(50);
                moveServo2(-50);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
        }

        if (adcThree() > 46)
        {
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(-50);
                moveServo2(50);
```

```
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
        }

        if (adcOne() > 35 && adcTwo() > 47)
        {
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(-100);
                moveServo2(-100);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);
                moveServo1(-50);
                moveServo2(50);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
        }

        if (adcOne() > 35)
        {
                moveServo1(75);
                moveServo2(50);
                ms_sleep(2000);
                moveServo1(-15);
                moveServo2(-15);
        }

        if (adcTwo() >47)
        {
```

```
                                        moveServo1(50);
                                        moveServo2(75);
                                        ms_sleep(2000);
                                        moveServo1(-15);
                                        moveServo2(-15);
                                }

        }

        moveServo1(25);
        moveServo2(25);
}

if ((PINA && 0b10000000)==0)
{
        align=1;
        moveServo1(-15);
        moveServo2(-15);
        do
        {
                moveServo1(25);
                moveServo2(-15);
        }while ((PINA && 0b01000000) != 0);
        moveServo1(-15);
        moveServo2(-15);
}

if ((PINA && 0b01000000)==0)
{
        align=1;
        moveServo1(-15);
        moveServo2(-15);
        do
        {
```

```
                moveServo1(-15);
                moveServo2(25);
        }while ((PINA && 0b10000000) != 0);
        moveServo1(-15);
        moveServo2(-15);
}

if (align == 1);
{
        moveServo3(100);
        ms_sleep(90000);
        moveServo3(0);

        moveServo4(100);
        ms_sleep(15000);
        moveServo4(0);


        PORTB = 0b00000010;
        ms_sleep(3000);

        moveServo3(-100);
        ms_sleep(30000);
        moveServo3(100);
        ms_sleep(30000);

        moveServo3(-100);
        ms_sleep(90000);
        moveServo3(0);

        moveServo1(-100);
        moveServo2(-100);
        ms_sleep(3000);
```

```
                moveServo1(-15);
                moveServo2(-15);
                ms_sleep(300);

                moveServo1(-40);
                moveServo2(25);
                ms_sleep(3000);

                moveServo1(-15);
                moveServo2(-15);

                align=0;
        }

        moveServo1(25);
        moveServo2(25);

        }
}
```