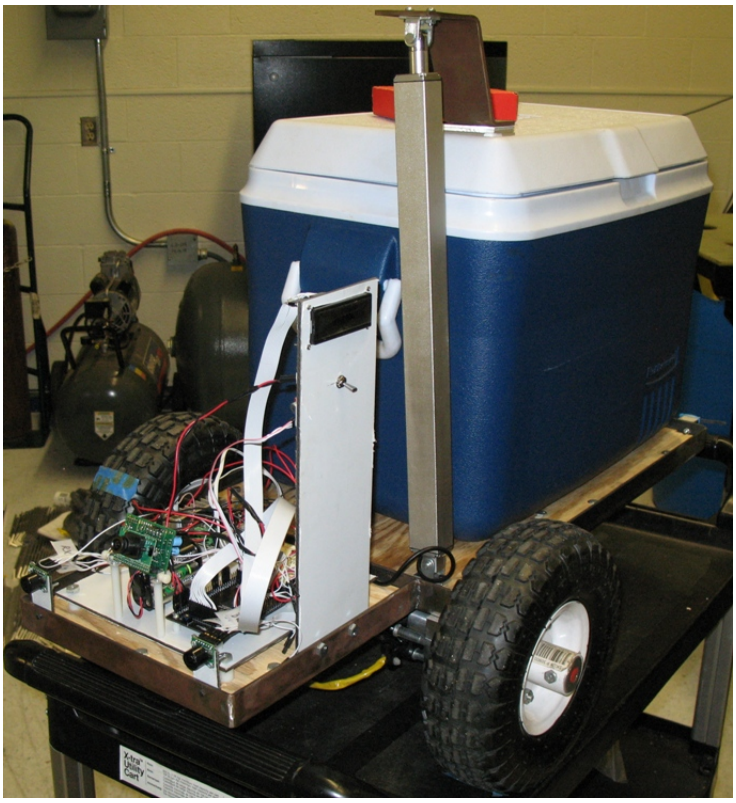
Tailgator's Best Friend

Final Report



Mathew Johnson

EEL 5666
Intelligent Machine Design Laboratory

Professors:
Dr. Arroyo
Dr. Schwartz

Teaching Assistants:
Mike Pridgen
Thomas Vermeer
Tim Martin
Ryan Stevens
Devin Hughes

Table of Contents

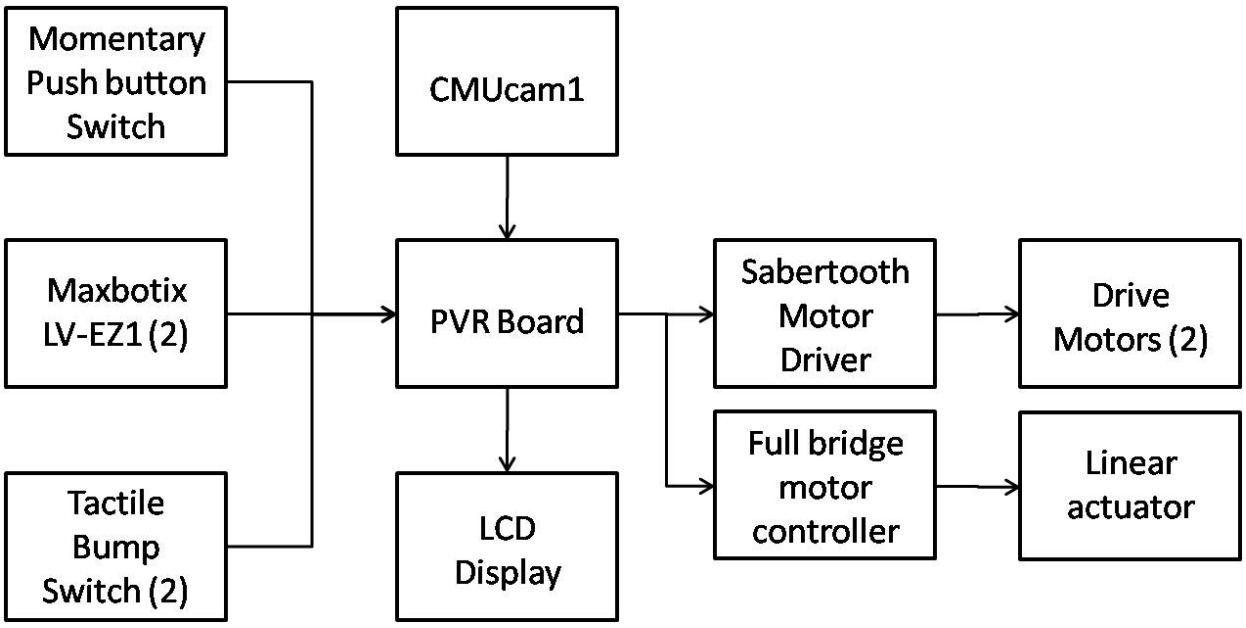
I. Abstract

This report details the specifications of Tailgator's Best Friend and how it functions. It is an autonomous robotic platform that carries a cooler. The platform is made from steel and plywood. It is driven by two DC motors powered by a 12V battery and the electronics are powered by a battery pack with 6 NiMH batteries. All components and behaviors are described herein.

II. Introduction

Coolers are an awkward, heavy item that nearly everyone has wrangled with one time or another. The purpose of Tailgator's Best Friend is to alleviate the difficulty of lugging one around. The weight of the cooler is supported by the robot instead of its owner. The Tailgator's Best Friend is made such that the owner merely has to walk while the cooler follows behind. Finally, when a beverage is desired from the cooler, a simple press of a button opens it.

III. Integrated System



IV. Mobile Platform

The platform is made of steel that was welded together. Steel was selected due to its high strength, ensuring it would carry the cooler load with ease. Also, steel is much less expensive than aluminum and easier to weld. Additionally, there is a separate housing for all of the sensitive electronics to ensure they are away from the cooler. This housing is located at the front of the robot to ensure there are many locations to mount sensors for tracking and obstacle avoidance.

Making the frame from steel turned out to be somewhat of a challenge. The steel needed to be welded and a steady holder was needed to hold the miter joints close together. The solution implemented was to machine four 90° grooves in wood blocks that were 0.75 inches deep that were the same thickness as the steel. Then, the steel pieces were slipped into the grooves and were held at perfect 90° angles while a small tack weld was done. By employing this method, the corners turned out to be quite square.

V. Actuation

The main drive motors of the robot are two Denso 12 volt electric motors. The motors were originally used as window lift motors and offer plenty of torque for this application. They output approximately 88 in-lb of torque each. The motors were originally equipped with a 7 tooth pinion gear that has been modified into a D shaft. The modification allowed for easy implementation of a direct drive system. The motors are coupled to the two front wheels, which are 10'' pneumatic tires. The rear of the robot is equipped with a 5 inch solid plastic caster that permits easy turns in any direction.

The drive system has been designed with two bearings mounted into a solid aluminum plate. The plate serves two purposes. First, it acts as a mount for the drive motor, to keep it solidly

attached to the robot. The motors are set to a specific offset from the mount using steel spacers. This was done to make sure there was sufficient room for the shaft coupler. Second, it acts as the housing to hold the bearings which support the 5/8 inch drive shaft. Each bearing is rated to carry over 100 lbs.

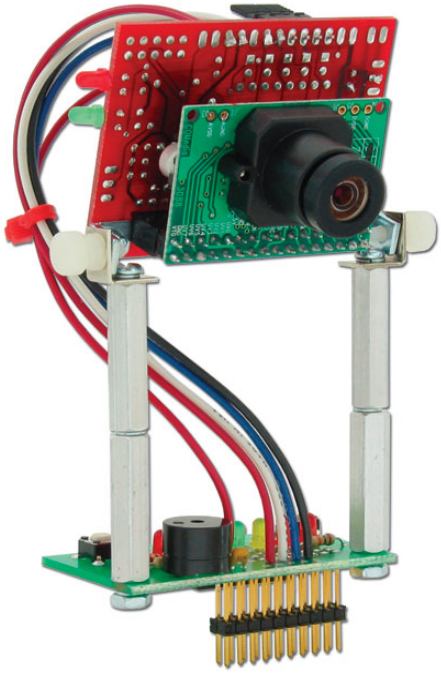
For connecting the output shaft from the motor to the drive shaft, a simple solid aluminum coupler was made. It was bored out to allow the 5/8 inch drive shaft to fit snuggly on one side and the 10 mm motor output shaft on the other. To rigidly connect the shafts to the coupler, the simplest method was used. A hole was drilled through the shaft and coupler on each side and a solid steel bolt was put through the holes. On the motor output shaft side, a 10-32 bolt was used and on the drive shaft side a ¼-20 was used.

The drive motors pull considerable current and require a 12v supply. To meet these needs, a 12v sealed lead acid absorbed glass mat (SLA AGM) battery was selected to drive them. The battery selected was an 18 Ah battery made by Universal Battery. It provides plenty of power to run the robot for a long period of time. As well, being an AGM battery, it can be positioned in any orientation and have no possibility of leaking acid.

For opening the cooler, a small linear actuator is employed. The actuator is 18.5 inch long when fully closed. It extends one foot to its fully open state at approximately 1.5 in/sec. The actuator is capable of pushing with 8 lbs of force. It is from Firgelli Automation. The actuator is controlled through a dual full bridge motor driver (L298N). Only half to the bridge is utilized and the circuit is built on a breadboard for easy debugging. The power for the actuator is 12V supplied from the SLA battery.

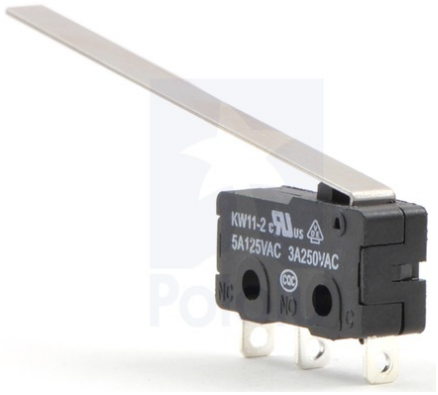
VI. Sensors

The main sensor that is being employed is the CMUcam1. This sensor is used to follow the user by tracking a led they will be wearing. As of right now, the BOE-BOT CMUcam1 is the sensor purchased. This sensor came without the proper electronics to enable RS-232 communication. Therefore, a Max232 chip and the required capacitors were purchased and populated on the board. The camera was focused using the Java GUI supplied on the CMUcam1 on a laptop.

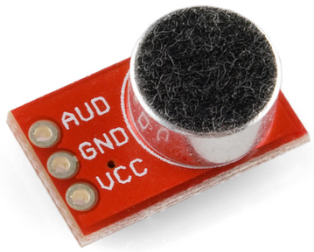


The obstacle avoidance is implemented using two types of sensors. First the robot has ultrasonic range finders to watch for obstacles in its path. The range finders selected are two Maxbotix LV-EZ1 sensors. These sensors have a fairly wide range they can detect objects. They are mounted on the front of the robot and overlap for a small area. They are chained together to ensure they are not interfering with each other and reliable data is acquired.

As an additional measure, the robot is outfitted with two tactile bump switches. The switches are located at the rear of the robot. They are located at the rear to make certain the robot does not back up into an obstacle. The sensors have a 50 mm arm attached to them which allow for most obstacles to be detected.



A microphone will also be implemented. This sensor is used to monitor for the sound of a clap signaling it is time to open or close the cooler. The sensor I have selected is the Breakout Board for Electret Microphone. It was selected because it was inexpensive and came equipped with a 100X opamp for amplifying sounds. Unfortunately, due to issues with the microphone, it was not implemented in the final robot. It was replaced with a momentary push button switch.



VII. Behaviors

The robot has a fairly straight forward set of behaviors. It moves based upon the location of the color the camera is tracking. That is, when the color being tracked is a pixel value below 15, the robot will make a hard right turn. When the value is above 15 but below 35, it will make a right turn. For pixel values between 35 and 45, the robot goes straight. When the pixel value is 45 to 65, the robot turns left. And for values above 65, it makes a hard left turn. To ensure the robot does not go off on its own, when the camera is not able to find the color it is tracking, the robot will stop.

The difference between a hard turn and a normal turn is a 5% difference in power sent to each motor. For regular turns, one motor receives 55% power and the other receives 60%. For hard turns one motor receives 50% power and the over receives 60%.

For obstacle avoidance and setting following distance, the robot uses the front mounted ultrasonic rangefinders. The robot will not move when the range finders return a value that corresponds to an object less than two feet in front of it, unless it is closer than one foot. At one foot, the robot will back away from the object to avoid collision.

On the rear of the robot, the two tactile bump switches act to ensure the robot does not back into anything. If the robot is backing up and impacts an object, the robot will slowly move forward.

Lastly, the cooler opening function is controlled with a momentary push button switch. When the switch is activated, the cooler will open. While the cooler is open, the robot does not move. When the switch is pressed again the cooler will close and the robot will again track its user.

VIII. Conclusion

Overall, I feel the project was a near success. The robot does follow the user; it avoids obstacles, and can open and close. The following system using the CMUcam1 is fairly good, but it is somewhat less reliable than I would like. Also, I am disappointed that I was unable to get the microphone working. That is the one item I would really like to get functional that I was unsuccessful with.

Given what I know now, looking back I would have done a few things differently. First, I should have become better friends with someone proficient in C much earlier on. I knew no C programming when I started this class and it made getting started quite difficult. Once I learned the basics I was able to get moving, but it took a lot of time and help from the TA's and fellow students.

It turned out to be a great plan to spend as much time making sure my drive system and frame were set early on. By doing this early on I could focus on the programming task. However, I wish I had selected components and ordered them much earlier than I did. Waiting until I was ready to work on items to order them, like the linear actuator, meant I was waiting for parts to arrive.

IX. Documentation

1) Sabertooth 2X25 motor driver manual, http://www.dimensionengineering.com/Sabertooth2X25.htm
2) L298N motor driver manual, www.mouser.com
3) PVR XMega128 Manual (Provided by Mike Pridgen and Thomas Vermeer with the PVR Board)
4) ATxmega128A1 manual and data sheet, www.atmel.com

X. Appendices

Robot Code

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

#define Sonar1 (ADCA6())                    //Define the variable Sonar as the
ADCA0 value
#define Sonar2 (ADCA7())
#define Mic (ADCA1())
int ADCA6(void);
int ADCA7(void);
int ADCA1(void);
int open = 0;
int val1 = 0;
int val2 = 0;
int check = 0;

void main(void)
{
        xmegaInit();                                    //setup XMega
```

```
	delayInit();                                    //setup delay functions
	ServoCInit();                                   //setup PORTC Servos
	ServoDInit();                                   //setup PORTD Servos
	ADCAInit();                                     //setup PORTA
analong readings
	lcdInit();                                      //setup LCD on
PORTK


	PORTH_DIR = 0x01;       //set bit 0 as output, 7..1 as input
	PORTH_OUT = 0;                  //set bit 0 as low
	PORTJ_DIR = 0x06;       //0 as input, 1 and 2 as out, rest as in
	PORTF_DIR = 0x0C;       //pin 2 and 3 as output
	PORTF_OUT = 0x0C;               //   and high

	PORTQ_DIR |= 1;                 //enable debug LED
	PORTQ_OUT &= ~(1);              //turn off debug LED

	//Initializaiton
	PORTE_DIR = PIN3_bm; //Pin 3 of port E is output
	PORTE_OUT = PIN3_bm; //Pin 3 of port E is TXO
	PORTE.DIRCLR = PIN2_bm; //Pin 2 of port E is RXO
	USARTE0_CTRLC = 0x03; // USART Control Register C: ASYNCHRONOUS, no
parity 1 stop bit 8 bit word
	USARTE0_BAUDCTRLA = 0x06; // Page 238 of Atmel manual, fbaud=9600 (my
case)=32MHz/(16*(((2^BSCALE) * BSEL)+1))
	USARTE0_BAUDCTRLB = 0xC1; // BSEL= 262 and BSCALE= -4 in 2s comp .
	USARTE0_CTRLB |= 0x08; // TX0 is on
	USARTE0_CTRLB |= 0x10; // RX0 is on
	static char *temp;

	//Program

	CMUsend("RS\r"); //Reset the camera

	delay_ms (5000); //Long enough to wait for ACK and to get the camera ready

	CMUsend("L1 1\r"); //To turn on green light

	temp=CMUreceive(); //Receive ACK
	delay_ms(5); //For some reason that I havent figure out, there must be a delay between
commands
	CMUsend("PM 1\r"); //Activate polling mode

	temp=CMUreceive(); //Receive ACK
	delay_ms(5);
```

```
CMUsend("mm 1\r");
delay_ms(5);


if(Mic > 4000)          //checks to make sure the mic is at 2.5 volts at startup
{
        check = 1;
}

while (1)
{
        CMUsend("TC 100 255 0 30 0 30\r"); //Track color
        temp=CMUreceive(); // Receive ACK
        temp=CMUreceive(); // Receive M packet
        delay_ms(5);

        lcdData(0x01);
        lcdString(temp); //LCD Write function

        int mx;
        int rdptr = 2;
        mx = temp[rdptr] - 0x30;
        rdptr++;
        if (temp[rdptr] != ' ')
        {
                mx = mx*10 + (temp[rdptr] - 0x30);
                rdptr++;
        }

        PORTF_OUTSET = 0x40;            //set bit 6 of port F high

        delay_us(200);                  //delay for 200us

        PORTF_OUTCLR = 0x40;            //set bit 6 of port F low

        char string[5];
        int i2 = sprintf(string,"%d", Mic);
        lcdGoto(1,0);
        lcdString(string);

        if(mx == 0 || Sonar1 < 500 || Sonar2 < 500 || open == 1)
        {
                ServoD4(0);
                ServoD5(0);
                lcdGoto(1,0);
                //lcdString("Stop");
```

```
                        if((Sonar1 < 350 || Sonar2 < 350) && open != 1)
                        {                                                                       //Loop
to avoid an obstacle that comes very close to the front of the robot
                                ServoD4(0);
                                ServoD5(0);

                                delay_ms(200);

                                int n;
                                for(n=0; n < 20; n++)
                                {
                                        //delay_ms(200);

                                        ServoD4(-50);
                                        ServoD5(-50);

                                        if(PORTJ_IN & 0x80 || PORTJ_IN & 0x40)
                                        {
        //Loop in case the robot backs into something
                                                ServoD4(0);
                                                ServoD5(0);

                                                delay_ms(200);

                                                ServoD4(30);
                                                ServoD5(30);

                                                delay_ms(500);

                                                n = 21;
                                        }
                                        delay_ms(50);
                                }
                        }
                }

                else if(mx < 15)
                {
                        ServoD4(60);
                        ServoD5(50);
                        lcdGoto(1,0);
                        //lcdString("Hard Right");
                }
                else if(mx > 15 && mx < 35)
                {
```

```
                ServoD4(60);
                ServoD5(55);
                lcdGoto(1,0);
                //lcdString("Turn Right");
        }
        else if(mx > 65)
        {
                ServoD4(50);
                ServoD5(60);
                lcdGoto(1,0);
                //lcdString("Hard Left");
        }
        else if(mx > 45 && mx < 65)
        {
                ServoD4(55);
                ServoD5(60);
                lcdGoto(1,0);
                //lcdString("Turn Left");
        }

        else if(mx > 35 && mx < 45)
        {
                ServoD4(60);
                ServoD5(60);
                lcdGoto(1,0);
                //lcdString("Go Straight");
        }

        val1 = PORTJ_IN & 0x01;             //set val1 to be the status of the button



        if(val1 == 0x01)
        {                                                       //if button is pushed (old
setup)
        //if(check == 1)                                        //checks to make sure mic is working
        //{
                //if(Mic < 2000)                                //if a clap is detected
                //{
                        ServoD4(0);                             //stop the robot
                        ServoD5(0);

                        PORTQ_OUT = 1;

                        if(open == 0)
```

```
                        {                                       //if the cooler is not
open
                                PORTH_OUT = 0x01;       //set bit 0 of H high which
turns on power
                                PORTJ_OUT = 0x02; //set bit 1 high and bit 2 low
                                PORTJ_OUTCLR = 0x04;    //set bit 2 low
                                PORTQ_OUT = 0;                  //Turn off LED

                                delay_ms(10000);                //delay 7 seconds to
allow cooler to open

                                PORTH_OUTCLR = 0x01;    //clear bit 0 of H which cuts
power
                                PORTQ_OUT = 1;                  //Tunr on LED

                                delay_ms(12000);        //delay to allow cooler to stay
open
                                if(val1 == 1 && val2 == 0)
                                {                               //only increment the
value if the current value is greater than the previous
                                open = 1;
                                }
                        }
                        else if(open == 1)
                        {                                                       //if the
cooler is open
                                PORTH_OUT = 0x01;                       //set bit 0 of H
high which turn on power
                                PORTJ_OUTCLR = 0x02;            //set bit 1 low
                                PORTJ_OUT = 0x04;               //set bit 2 high
                                PORTQ_OUT = 0;                          //Turn off
LED

                                delay_ms(11000);                //delay 9 seconds to
allow cooler to fully close

                                PORTH_OUTCLR = 0x01;    //clear bit 0 of H which cuts
power
                                PORTQ_OUT = 1;                  //Turn on LED
                                if(val1 == 1 && val2 == 0)
                                {                                       //only
increment the value if the current value is greater than the previous
                                open = 0;
                                }
                        }
                }
```

```
                //}
        }

        val2 = val1;                            //set val2 to be the previous state of the button

}
```

PVR.c Code Modifications for CMUcam1 (Thanks to Alexis Mesa)

```
void CMUsend(char *command)
{
        int i = 0;
        //lcdString("1");
        while (command[i] != '\0') //While command does not end do...
        {
        //lcdString("2");                                                       // 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)));
        //      lcdChar(command[i]);
                i++;
        }

}

char CMUreceive(void)
{
int i = 0;

        static char *data;
        do
        {
        //lcdString("1");                                                       // 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
        //lcdString("2");
        return data;
}
```