Formal Report

MODS (Mobile Office Delivery System)

Kevin Hoffman

EEL5666

Dr. Arroyo, Dr. Schwartz

Tim Martin, Ryan Stevens

Table Of Contents

Abstract

The world is progressively getting more and fast paced. People have this level of expectation concerning how quickly they can acquire goods. One of the largest companies, Wal-Mart, is a business that boomed with a model of having everything one may need under one roof. Fast food chains are everywhere and show no sign of shrinking. America is a country where speed is everything. The MODS robot is an autonomous system that will take the load of office delivery and quickly optimize it by removing the human factor and allow those resources to be extended elsewhere.

MODS is a system that will be utilized to transport goods between rooms in any building amongst any floors without fear of obstacles. In the full system there will be two additional groups that will travel by air and outside land-nav. For the purpose of this project only a single robot will travel between two rooms on a single floor with little to no obstacle detection. After pickup and delivery MODS will return back to its point of origin

The robot will be constructed along the lines of a Modular approach where all parts should work by themselves. Most of the equipment is off the shelf sensors and mechanics with the exception of a special navigation system. The obstacle avoidance is conducted via four wide band sonars. Navigation is conducted by a camera where all image processing is handled onboard.

Introduction

The problem is straight forward. An autonomous delivery system will allow movement of packages relieving the stress, adding function, and improving efficiency to the current system of delivery service.

The objective of this project is to create such a system specially designed to operate within the confines of an office building. The two most important factors concerning this project are completion of delivery and safety of package while in route. In theory it should be able to traverse floors and operate without fear of obstacles that might crush the machine. In practice and prototype only a single floor and limited obstacles is overcome. There is very little in terms of package handling and protection.

The rest of this document will describe in a how the machine is broken down and how every system will interact and fit into the overall design of MODS. The navigation is produced using a single camera with all image processing conducted on board and sent to the Epiphany DIY for motor control. Obstacle avoidance is produced using four separate sonars in 360 degree coverage. With these two systems it is possible to complete the design concept for this application.

Executive Summary

MODS is an interesting creature. She accomplishes her main task and has successfully picked up and delivered an object. She successfully obstacles avoids in a very close 360 coverage area. At this point she can only go to a single pickup location, which is located inside a room, and move to a single delivery location which is located in a hallway. After this she can successfully return to her starting location. The way that she is programmed though she should be able to just rearrange the course and add and subtract waypoints to go to any point desired. This has yet to be tested.

MODS turned out to be a much more difficult project than what was first imagined. Since I had never dealt with image processing it took more time and effort to get that working than any other part of the entire project. It still isn't even up to the expectation that I wanted to accomplish.

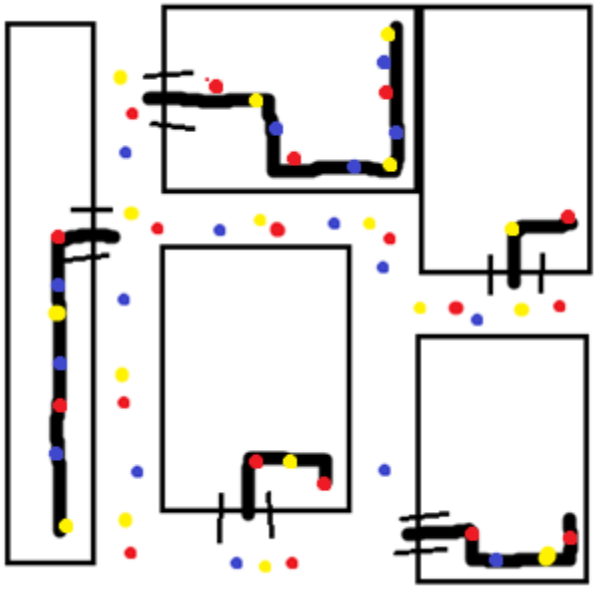
Integrated System

MODS (Mobile Office Delivery System)

The office building that each robot is to support will utilize a waypoint system to guide the robot from point to point. This waypoint system will include all areas of a buildings including the rooms that they will be delivering into . Colored pieces of paper will be on the ground in a certain sequence that will guide the robot to its intended destination. Inside the rooms a continuation of the paper sequence following path will be constructed that will slowly guide the robot to the rooms station. The paper sequence in the room will be at a shorter interval to ensure quick motion. For the purpose of this demonstration only a single room and only one point of pickup and delivery will exist Once the robot reaches its point of pick and point of delivery it will open its cargo bay door and wait for a package to be put inside. A button on the rear will be pressed by human interaction to tell the robot to proceed on its way. There will be limited obstacle avoidance inside the room due to the fact that rooms will most likely have little room for movement and obstacle avoidance will confuse MODS. Outside the room sonars will be deployed to detect proximity of the moving obstacles. If a certain threshold is violated the robot will move off path until threshold is met again.

When all of these systems are working alongside each other the system will be complete and any building can theoretically be programmed (i.e. if two rooms can operate then any number of rooms can operate).

The original design included line following for in room navigation. It proved too difficult to integrate into the board due to how the microcontroller handled it’s pin functions and camera navigation took its place.

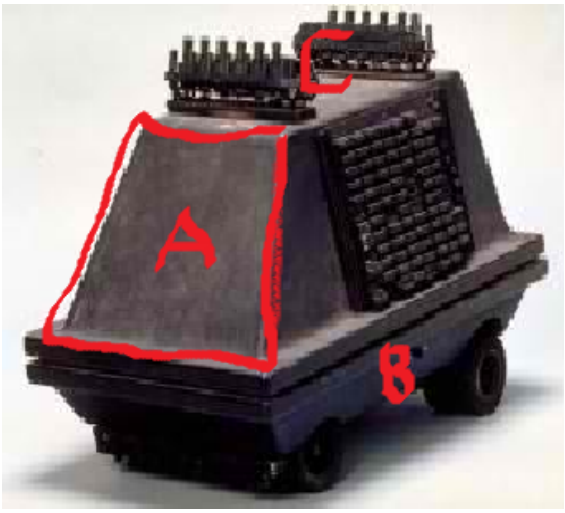


Overall design of the System

Mobile Platform

The platform's inspiration is based on the MSE-6 Repair Droid from the Star Wars franchise. Completely made of single layer copper plating PCB this robot will be driven with two powerful motors. A single caster wheel is located at the front center of the robot. A main bay exist that will hold the package. A single bay door at the front of the robot will open up to load and unload the package. Above and below the main bay two additional support areas exist to house the sensors, control boards, and additional daughter boards. In the future these areas will be covered and put in a housing. The wheels will intrude into the lower bay support rooms so to lower the center of gravity of the whole robot.

The caster wheel was a real nightmare. Due to all the small designs that were purchased and the way they were ball bearings on ground they were getting very dirty and I had to use Swiffer cleaners to keep them from clogging up.



MSE-6 Repair Droid

A is the storage bay door to main bay. B is the lower support bay. C is the upper support bay

Actuation

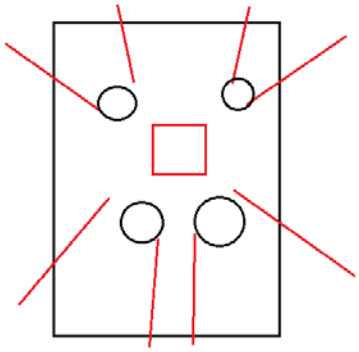
MODS will be operated by two power 12V motors. These motors are capable of, at the rated voltage, 200rpms and 180 ounces per inch of torque. Their stall current is 5A. They are geared 50:1 which turned out to be plenty enough to move the robot approx. 14-20 inches per second at 12V. This is all dependent on the final weight. The weight at the final presentation this robot can accomplish this speed at only ¾ power. Originally there was supposed to be servos controlling everything including the caster wheel and camera up front. This proved to be a useless design and even though the servos were mounted they were never used and were removed. The only problem with the motors were that they were so powerful that they refuse to move at small speeds and the robot could not do fine tuned movements.

Motor and Servo in use

Sensor

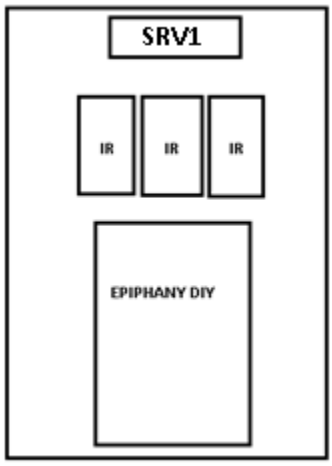
MODS are equipped with a wide array of sensors. In the top support bay 4 ultrasonic sensors (two front, two rear) are positioned to detect obstacles coming within three threshold values. These sonars have a wide beam angle and a 6” resolution past 15”. The three threshold values are safe, avoid, and move. Safe is normal operation, avoid is obstacle avoidance to a safe distance, and move is a detection that if an obstacle is so close to it’s rear that avoidance is not likely then all movement systems will overdrive and the motors will operate at full power getting the robot out of the way. These systems worked pretty effectively

In addition to the sonars in the top support bay a 3D compass module will help navigate the robot. Being able to detect if MODS is on the right bearing will give an extra degree of precision for an already complex task. The compass is also equipped with a tilt sensor and accelerometer so in case the robot is kicked over or pushed over it can send out a sound alert. The compass was mounted but it was never used since camera navigation kept the robot on it’s path. It was then removed to make way for a flashlight which was also removed.



Top bay where circles are Sonars and square is compass

In the lower support bay a blackfin SRV-1 camera is positioned to detect colored paper for navigation purposes. A sequence will be programmed into the MODS and the SRV-1 will "blob detect" the color sequence that guides it to the room it needs to be in. Located at the bottom are three IR sensors that were never removed. These will most likely removed at some point. They were originally used to line following upon entry into a room but were taken out of the design and the replaced by camera navigation.



Lower Support Bay

All of these sensors are ultimately connected to the Epiphany DIY controller board.

Behavior

The behavior of the robot is very erratic. It is a joke amongst my family that you should never mention the word P-R-O-O-F and hide all cameras when she is running because she never likes to work when a camera is out. This also is well defined when she decided to corrupt her camera firmware on pre-demo day and then she fried on her own on actual demo day. It will be interesting to see what she does before Media Day. Other than these issues she works for the most part in a well lit environment. She works even better when she has to detect very small colors as opposed to large pieces of paper. I will admit this is different from what I imagined what would happen.

Experimental Layout

No real experiments were conducted in this project. The majority of all the time went into calibration of the camera. The only experiment was the threshold values required of the sonars. This was conducted by taking a sheet of paper and moving it to the range required and reading off the LCD screen what the ADCs were reading back. These values were then set into the program and that was concluded. The camera calibration was and still is a nightmare to deal with. It still isn't completed at this point in the project. The camera is to detect three different colors. The camera retrieves images in the YUV spectrum. Thus luminance is a big issue and lighting in the environments are proving to be a hassle. They cause the robot to not see the colors required of them. So in every environment I have to test the brightness in the room and major points of light distribution. Some of these problems were solved with a flashlight and brighter lights. It is unknown how well they will perform outside in the natural sunlight.

Conclusion

MODS is a beast in every sense of the term. She is built like a tank and moves fairly quickly and fairly loudly. She is programmed to track three different colors decently. She has absolutely no clue how to differentiate between color noise in the background and even confuses white and black for the colors that she is tracking. With little to no experience in image processing one semester proved not enough time to do what she needed in this regard. Nothing exceeds expectation but she will do as she is suppose to do. She has successfully navigated between a single room with pickup and delivered to a single point and returned to her point of origin. Everything can be improved and nothing is what I would consider acceptable. There just wasn't enough time in the semester and money in bank for everything to work properly. For students looking into this work I would strongly avoid using this camera. It is not a bad camera and works like a charm when she wants to but she is too advance and her company has absolutely no customer support. The frustration level is off the chart in this respect.

In all honest if I were to start over I would never use a camera because it is just too frustrating in the limited time that you have. If I had to use the camera then I would start with making sure the company is there to back you up. I got screwed with my first company (Seattle Robotics) and I got screwed with my current company (Surveyor Corporation). Make sure you have support behind you. I'm still waiting on a board to be delivered and I will be looking for my money back very soon. Start early and start fast and make sure you are taking not a single other class because this will definitely ruin your semester.

Appendices

CODE

```
/**

* \file

*

* \brief Empty user application template

*

*/

/*

* Include header files for all drivers
that have been imported from

* AVR Software Framework (ASF).

*/

#include

<asf.h>

#include

<avr/io.h>

#include

<ctype.h>

#include

<stdint.h>

#include

<stdio.h>

#include

<util/delay.h>

#include

"motor.h"

#include

"lcd.h"

#include

"uart.h"

#include

"RTC.h"

#include

"picServo.h"

#include

"ADC.h"

#include

"switch.h"

#include

"sonarRX.h"

#define

DbLedOn() (PORTR.OUTCLR = 0x02) //Turns
the debug led on. The led is connected
with inverted logic

#define

DbLedOff() (PORTR.OUTSET = 0x02) //Turns
the debug led off. The led is connected
with inverted logic

#define

DbLedToggle() (PORTR.OUTTGL = 0x02)
//Toggles the debug led off. The led is
connected with inverted logic

#define

ADC_COUNTER 100 //Countdown timer till
ADC values are read

//Method only in use because Timer0 is
too fast for use

#define

ADC0_THRES 282

#define

ADC1_THRES 70

#define

ADC2_THRES 243

#define

ADC3_THRES 244

#define

RED 0

#define

BLUE 2

#define

PURPLE 3

//#define GREEN 1

/***********************************/

//GLOBAL VARIABLES GO HERE

char

uartD0_rx[256];

int

color_center = 0;

int

spinning_flag = 1;

int

x1 = 0;

int

x2 = 0;

int

counter = 0;

//int color_pattern[2]; //LIST OF ORDER
TO FOLLOW

int

obstacle_detect = 0; //Zero = no
detection. One = Detection

int

overshoot = 0;

int
```

```
hold_color = 0;

int

ctemp = 0;

char

route[11] = {RED,BLUE,RED,PURPLE,RED,BLU
E,RED,BLUE,RED,BLUE,RED};

char

turn_seq[11] = {0,0,0,1,0,1,0,0,0};

int

route_counter = 1;

int

turn_counter = 0;

int

route_counter_size = 11;

int

pickup = 4;

int

delivery = 7;

/***********************************/

void

parse_loc()

{

////////////////////

stdout = &lcd_str;

////////////////////

//printf("PARSE");

int loc[3];

int counter = 0;

for(int k = 0; k < 128; k++)

{

if(uartD0_rx[k] == '-') //k is now the
position of the '-' in memory

{

for(int j = 1; j < 10; j++)

{

if(uartD0_rx[k+j] == ' ') //blank space

{

loc[counter] = k+j;

counter++;

}

}

//printf("CALCX2X1");

//_delay_ms(3000);

if((loc[2] -
 loc[1] == 4)){x2 = ((uartD0_rx[(loc[1]+
1)] -
 48)*100) + ((uartD0_rx[(loc[1]+2)] -
 48)*10) + ((uartD0_rx[(loc[1]+3)] -
 48)*1);

printf("Px2: %c%c%c
",uartD0_rx[(loc[1]+1)],uartD0_rx[(loc[1
]+2)],uartD0_rx[(loc[1]+3)]);

}

if((loc[2] -
 loc[1] == 3)){x2 = ((uartD0_rx[(loc[1]+
1)] -
 48)*10) + ((uartD0_rx[(loc[1]+2)] -
 48)*1);

printf("Px2: %c%c
",uartD0_rx[(loc[1]+1)],uartD0_rx[(loc[1
]+2)]);

}

if((loc[2] -
 loc[1] == 2)){x2 = ((uartD0_rx[(loc[1]+
1)] - 48)*1);

printf("Px2: %c
",uartD0_rx[(loc[1]+1)]);

}

if((loc[1] -
 loc[0] == 4)){x1 = ((uartD0_rx[(loc[0]+
1)] -
 48)*100) + ((uartD0_rx[(loc[0]+2)] -
 48)*10) + ((uartD0_rx[(loc[0]+3)] -
 48)*1);

printf("Px1: %c%c%c
",uartD0_rx[(loc[0]+1)],uartD0_rx[(loc[0
]+2)],uartD0_rx[(loc[0]+3)]);

}

if((loc[1] -
 loc[0] == 3)){x1 = ((uartD0_rx[(loc[0]+
1)] -
 48)*10) + ((uartD0_rx[(loc[0]+2)] -
 48)*1);

printf("Px1: %c%c
",uartD0_rx[(loc[0]+1)],uartD0_rx[(loc[0
]+2)]);

}

if((loc[1] -
 loc[0] == 2)){x1 = ((uartD0_rx[(loc[0]+
1)] - 48)*1);

printf("Px1: %c
",uartD0_rx[(loc[0]+1)]);

}

////////////////////////////////

LCDCommand(LCD_CLEAR);

_delay_ms(100);

////////////////////////////////

break;

}

}

}

void

zero_uartD0()

{

uint16_t n = 0;

while(n < 256) //zero out the array

{

uartD0_rx[n] = 0;

n++;

}

}

calibrate_cbins()

{

stdout = &SRV1_str; //set resolution to
320x240

printf("b");
```

```
_delay_ms(1000);

stdout = &SRV1_str;

printf("vc0000200000140165255"); //Red 0

_delay_ms(1000);

stdout = &SRV1_str;

printf("vc2000130150255000110"); //Blue
2

_delay_ms(1000);

stdout = &SRV1_str;

printf("vc3000160130255140255");
//Purple 3

_delay_ms(1000);

stdout = &SRV1_str;

printf("vc1000150000145000125"); //Green
4

_delay_ms(1000);

stdout = &lcd_str;

LCDCommand(LCD_CLEAR);

_delay_ms(1000);

}

poll_calc_red()

{

zero_uartD0();

stdout = &SRV1_str;

counter = 0;

printf("vb0");

_delay_ms(50);

/////////////////////

stdout = &lcd_str;

LCDCommand(LCD_CLEAR);

_delay_ms(25);

/////////////////////

parse_loc();

color_center = ((x2 - x1) / 2) + x1;

/////////////////////////

stdout = &lcd_str;

printf("x1: %d x2: %d rc:
%d", x1, x2, color_center);

_delay_ms(50);

LCDCommand(LCD_CLEAR);

_delay_ms(25);

/////////////////////////

}

poll_calc_blue()

{

zero_uartD0();

stdout = &SRV1_str;

counter = 0;

printf("vb2");

_delay_ms(50);

////////////////////////

stdout = &lcd_str;

LCDCommand(LCD_CLEAR);

_delay_ms(25);

////////////////////////

parse_loc();

color_center = ((x2 - x1) / 2) + x1;

///////////////////////

stdout = &lcd_str;

printf("x1: %d x2: %d rc:
%d", x1, x2, color_center);

_delay_ms(50);

LCDCommand(LCD_CLEAR);

_delay_ms(25);

///////////////////////

}

poll_calc_purple()

{

zero_uartD0();

stdout = &SRV1_str;

counter = 0;

printf("vb3");

_delay_ms(50);

/////////////////////

stdout = &lcd_str;

LCDCommand(LCD_CLEAR);

_delay_ms(25);

///////////////////////

parse_loc();

color_center = ((x2 - x1) / 2) + x1;

///////////////////////

stdout = &lcd_str;

printf("x1: %d x2: %d rc:
%d", x1, x2, color_center);

_delay_ms(50);

LCDCommand(LCD_CLEAR);

_delay_ms(25);

////////////////////////

}

/*poll_calc_green()

{

zero_uartD0();

stdout = &SRV1_str;

counter = 0;

printf("vb1");

_delay_ms(50);

stdout = &lcd_str;
```

```
LCDCommand(LCD_CLEAR);

_delay_ms(25);

parse_loc();

color_center = ((x2 - x1) / 2) + x1;

stdout = &lcd_str;

printf("x1: %d x2: %d rc: %d", x1, x2,
color_center);

_delay_ms(50);

//LCDCommand(LCD_CLEAR);

_delay_ms(25);

}*/



void

open_door()

{

for(int i = 140; i > 40; i--)

{

setServoAngle(i,1);

_delay_ms(10);

}

_delay_ms(1000);

}

void

close_door()

{

for(int i = 40; i < 140; i++)

{

setServoAngle(i,1);

_delay_ms(10);

}

_delay_ms(1000);

}

void

poll_color(int bin_num)

{

//Zero out all values for calculations

color_center = 0;

x1 = 0;

x2 = 0;

uint8_t focuser = 0;

uint8_t temp = 0;

if(bin_num == 0) //bin 0 is red bin

{

while(focuser < 2)

{

poll_calc_red();

if(color_center > 0)

{

temp++;

}

focuser++;

}

if(temp < 2)

{

color_center = 0;

}

}

else if(bin_num == 2) //bin 2 is blue
bin

{

while(focuser < 2)

{

poll_calc_blue();

if(color_center > 0)

{

temp++;

}

focuser++;

}

if(temp < 2)

{

color_center = 0;

}

}

else if(bin_num == 3)

{

while(focuser < 2)

{

poll_calc_purple();

if(color_center > 0)

{

temp++;

}

focuser++;

}

if(temp < 2)

{

color_center = 0;

}

}

////////////////////////////////////
////////

if((x2 <= 80) || (x1 >= 240))

{

color_center = 0;

}
```

```
if(color_center > 0)

{

hold_color = 0;

}

if(spinning_flag == 1)

{

if(color_center < 100 || color_center > 220)

{

color_center = 0;

}

}

//Set conditional for changing looking for other colors

/*if(spinning_flag == 0)

{

stdout = &lcd_str;

printf("SPINNING FLAG");

_delay_ms(100);

}

if(color_center == 0)

{

stdout = &lcd_str;

printf("COLOR FLAG");

_delay_ms(100);

}*/

if((spinning_flag == 0) && (color_center == 0) && (hold_color == 0))

{

overshoot = 1;

//stdout = &lcd_str;

//printf("CHANGE");

//_delay_ms(100);

if((route_counter == pickup) || (route_counter == delivery))

{

setMotorDuty(2,256,MOTOR_DIR_FORWARD_gc);

setMotorDuty(4,256,MOTOR_DIR_FORWARD_gc);

_delay_ms(1000);

stdout = &lcd_str;

printf("PICKUPDELIVERY");

_delay_ms(2000);

open_door();

flipHTSstatus(0);

switch_init();

_delay_ms(200);

while(!returnHTSstatus()){}

flipHTSstatus(0);

switch_unit();

cli();

motorInit();

_delay_ms(5000);

close_door();

_delay_ms(3000);

cli();

PMIC.CTRL |= PMIC_LOLVLEN_bm;

sei();

}

if(route_counter <= route_counter_size)

{

ctemp = route[route_counter];

route_counter++;

}

}

if(route_counter > route_counter_size)

{

setMotorDuty(2,0,MOTOR_DIR_NEUTRAL_gc);

setMotorDuty(4,0,MOTOR_DIR_NEUTRAL_gc);

stdout = &lcd_str;

printf("COMPLETE SEQUENCE");

while(1){;}

}

}

void

move_to_target() //MOTOR 4 is on Robot Right. MOTOR 2 is on Robot Left.

{

if((spinning_flag == 1) && (color_center > 0)) //that the robot is currently in spinning phase and color is detected

{

//stdout = &lcd_str;

//printf("SPIN TO RUN");

//_delay_ms(100);

spinning_flag = 0; //Reset flag

setMotorDuty(2,0,MOTOR_DIR_NEUTRAL_gc);

setMotorDuty(4,0,MOTOR_DIR_NEUTRAL_gc);

_delay_ms(50);

}

if((color_center >= 1) && (color_center < 80)) //target is on far left. move quick left to get to target

{

setMotorDuty(2,786,MOTOR_DIR_FORWARD_gc); //LEFT

setMotorDuty(4,820,MOTOR_DIR_FORWARD_gc); //RIGHT

}

else if((color_center >= 80) && (color_center < 150)) //target is on the left. move shallow left to get to target
```

```
{

setMotorDuty(2,771,MOTOR_DIR_FORWARD_gc)
; //LEFT

setMotorDuty(4,755,MOTOR_DIR_FORWARD_gc)
; //RIGHT

}

else if((color_center >= 150) && (color_
center < 170)) //target is in the
center. proceed on course

{

setMotorDuty(2,786,MOTOR_DIR_FORWARD_gc)
; //LEFT

setMotorDuty(4,730,MOTOR_DIR_FORWARD_gc)
; //RIGHT

}

else if((color_center >= 170) && (color_
center < 240)) //target is on the right.
move shallow right to get to target

{

setMotorDuty(2,796,MOTOR_DIR_FORWARD_gc)
; //LEFT

setMotorDuty(4,720,MOTOR_DIR_FORWARD_gc)
; //RIGHT

}

else if(color_center >= 240) //target is
on the far right. move quick right to
get to target

{

setMotorDuty(2,786,MOTOR_DIR_FORWARD_gc)
; //LEFT

setMotorDuty(4,700,MOTOR_DIR_FORWARD_gc)
; //RIGHT

}

else

{

cli();

//stdout = &lcd_str;

//printf("SPINNING");

//_delay_ms(3000);

spinning_flag = 1;

if(overshoot == 1)

{

setMotorDuty(2,176,MOTOR_DIR_FORWARD_gc)
;

setMotorDuty(4,128,MOTOR_DIR_FORWARD_gc)
;

_delay_ms(250);

setMotorDuty(2,32,MOTOR_DIR_BACKWARD_gc)
;

setMotorDuty(4,32,MOTOR_DIR_BACKWARD_gc)
;

_delay_ms(200);

overshoot = 0;

}

else

{

if(turn_seq[route_counter-1] == 0)

{

setMotorDuty(2,512,MOTOR_DIR_FORWARD_gc)
; //LEFT

setMotorDuty(4,512,MOTOR_DIR_BACKWARD_gc
); //RIGHT

_delay_ms(70);

setMotorDuty(2,128,MOTOR_DIR_FORWARD_gc)
; //LEFT

setMotorDuty(4,128,MOTOR_DIR_BACKWARD_gc
); //RIGHT

_delay_ms(200);

}

else

{

setMotorDuty(4,512,MOTOR_DIR_FORWARD_gc)
; //LEFT

setMotorDuty(2,512,MOTOR_DIR_BACKWARD_gc
); //RIGHT

_delay_ms(70);

setMotorDuty(4,128,MOTOR_DIR_FORWARD_gc)
; //LEFT

setMotorDuty(2,128,MOTOR_DIR_BACKWARD_gc
); //RIGHT

_delay_ms(200);

}

}

sei();

}

x1 = 0;

x2 = 0;

color_center = 0;

}

void

run_ob_avoid()

{

uint16_t ADCA0_val = 0;

uint16_t ADCA1_val = 0;

uint16_t ADCA2_val = 0;

uint16_t ADCA3_val = 0;

ADCA_request(0,1); //FRONT LEFT

if(ADCA_CH0_ConvComplete)

{

//stdout = &lcd_str;

ADCA0_val = ADCA_getVal(0);

//printf("ADC0: %u", ADCA0_val);

if(ADCA0_val <= ADC0_THRES) //threshold
value approximately 18 inches to 2 feet
from sonar

{

obstacle_detect = 1;

hold_color = 1;

//printf("Obstacle Detected");

setMotorDuty(2,806,MOTOR_DIR_FORWARD_gc)
; //LEFT
```

```
setMotorDuty(4,600,MOTOR_DIR_FORWARD_gc)
; //RIGHT

}

}

_delay_ms(25);

ADCA_request(1,0); //FRONT RIGHT

if(ADCA_CH1_ConvComplete)

{

//stdout = &lcd_str;

ADCA1_val = ADCA_getVal(1);

//printf("ADC1: %u", ADCA1_val);

if(ADCA1_val <= ADC1_THRES) //threshold
value approximately 18 inches to 2 feet
from sonar

{

obstacle_detect = 1;

hold_color = 1;

//printf("Obstacle Detected");

setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc)
; //LEFT

setMotorDuty(4,880,MOTOR_DIR_FORWARD_gc)
; //RIGHT

}

}

_delay_ms(25);

ADCA_request(2,3); //BACK LEFT

if(ADCA_CH2_ConvComplete)

{

//stdout = &lcd_str;

ADCA2_val = ADCA_getVal(2);

//printf("ADC2: %u", ADCA2_val);

if(ADCA2_val <= ADC2_THRES) //threshold
value approximately 18 inches to 2 feet
from sonar

{

obstacle_detect = 1;

hold_color = 1;

//printf("Obstacle Detected");

setMotorDuty(2,826,MOTOR_DIR_FORWARD_gc)
; //LEFT

setMotorDuty(4,740,MOTOR_DIR_FORWARD_gc)
; //RIGHT

}

}

_delay_ms(25);

ADCA_request(3,2); //BACK RIGHT

if(ADCA_CH3_ConvComplete)

{

//stdout = &lcd_str;

ADCA3_val = ADCA_getVal(3);

//printf("ADC3: %u", ADCA3_val);

if(ADCA3_val <= ADC3_THRES) //threshold
value approximately 18 inches to 2 feet
from sonar

{

obstacle_detect = 1;

hold_color = 1;

//printf("Obstacle Detected");

setMotorDuty(2,826,MOTOR_DIR_FORWARD_gc)
; //LEFT

setMotorDuty(4,840,MOTOR_DIR_FORWARD_gc)
; //RIGHT

}

}

_delay_ms(25);

//LCDCommand(LCD_CLEAR);

//_delay_ms(100);

if(((ADCA0_val > ADC0_THRES && ADCA1_val
 > ADC1_THRES) && ADCA2_val > ADC2_THRES
) && ADCA3_val > ADC3_THRES)

{

setMotorDuty(2,256,MOTOR_DIR_FORWARD_gc)
;

setMotorDuty(4,256,MOTOR_DIR_FORWARD_gc)
;

//printf("NO DETECT");

obstacle_detect = 0;

//spinning_flag = 1;

LCDCommand(LCD_CLEAR);

_delay_ms(100);

printf("%d", ctemp);

_delay_ms(500);

}

}

////////////////////////////////////////
////////////////////////////////////////
////////////////////////////////////////

int

main (void)

{

/*****************************/

/*****************************/

//All initializers methods located here

cli();

board_init(); /*This function originates
in the file init.c, and is used to
initialize the Epiphany DIY

motorInit() is declared within because
by default you the user should define
what your

motor setup is to prevent hurting the
Epiphany. You can do this by

*/

cli();

//Suspended bootloader LED

DbLedOn(); //I like to do this by
default to show the board is no longer
suspended in the bootloader.
```

```
//uartInit(&USARTC0,57600); /*as can be seen in the schematic. This uart is connected to the USB port.

// This function initializes this uart*/

uartInit(&USARTD0,115200); //UART initialization for the SRV-1 on port D0 at 115200 baud

motorInit(); //Initializes the motors

cli();

servoControlInit(); //Initializes the Servos

cli();

ADCsInits(); //this function initializes the ADCs inside the Xmega

cli();

LCDInit(); //Initializes the LCD Screen

cli();

//sonarRX_init(); //Initializes the sonarRX

cli();

DbLedOff(); //Light Sequence to indicate Initialization finished

_delay_ms(1000);

stdout = &lcd_str;

printf("Execution Ready\n");

DbLedOn();

_delay_ms(500);

DbLedOff();

_delay_ms(500);

printf("3..");

DbLedOn();

_delay_ms(500);

DbLedOff();

_delay_ms(500);

printf("2..");

DbLedOn();

_delay_ms(500);

printf("1..");

DbLedOn();

_delay_ms(500);

DbLedOff();

_delay_ms(500);

printf("GO");

for(uint8_t k = 0; k < 10; k++)

{

DbLedOn();

_delay_ms(100);

DbLedOff();

_delay_ms(100);

}

LCDCommand(LCD_CLEAR);

//END ALL INITIALIZING METHODS

/*****************************/

//ZERO OUT UART STORAGE ARRAY

zero_uartD0();

/*****************************/

// Enable low interrupt level in PMIC and enable global interrupts. Begin all Interrupts

PMIC.CTRL |= PMIC_LOLVLEN_bm;

sei();

/*****************************/

//MAIN WHILE LOOP - PERMANENT EXECUTION

//CURRENT EXECUTION LIST

/*

//- Board, UART, Motor, Servo, ADC, Switch, LCD, SonarRx - ALL INITIALIZED IN THIS ORDER

*/

_delay_ms(4000);

calibrate_cbins();

close_door();

stdout = &lcd_str;

sei();

uint16_t i,j ;

while(1)

{

//setServoAngle(140,1);

//run_ob_avoid();

//if(obstacle_detect == 0)

//{

//poll_color(ctemp);

//move_to_target();

//}

//*************************************************************

//Test code for camera calibration

//NOTE: To get proper reading then put directly in front of camera

//*************************************************************

stdout = &SRV1_str;

printf("vp01600120");

_delay_ms(1000);

stdout = &lcd_str;

LCDCommand(LCD_CLEAR);

_delay_ms(1000);

}

}

//END OF MAIN STATEMENT

/*****************************/

/*****************************/

//Additional functions go here
```

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

//Interrupt Subroutines located here

//IMPORTANT NOTE: cli() at beginning of
ISR and sei() at end of ISR

ISR(USARTD0_RXC_vect)

{

cli();

stdout = &lcd_str;

while(USARTD0.STATUS & USART_RXCIF_bm)

{

if(USARTD0.STATUS & USART_BUFOVF_bm)

{

//printf("BUFFER OVERFLOW");

char temp = USARTD0.DATA;

break;

}

uartD0_rx[counter] = USARTD0.DATA;

if(uartD0_rx[counter] == '\r')

{

uartD0_rx[counter] = 0;

}

if(uartD0_rx[counter] == '\n')

{

uartD0_rx[counter] = 0;

}

printf("%c",uartD0_rx[counter]);

counter++;

}

}

sei();

}

ISR(BADISR_vect)

{

}

//END ALL INTERRUPT SUBROUTINES
```