Mobile Robotics Final Exam

Charlie Coleman

Question 3

Straight Wall

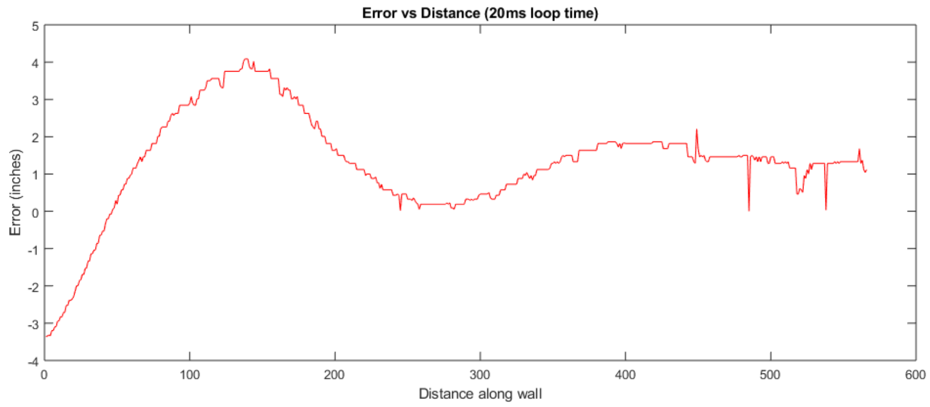
For the straight wall portion of this exam, a simple PD controller was used. Through trial and error, Kp & Kd values of 0.3 and 9.0 (respectively) were found to give a good result. Loop time was controlled using the onboard timer. A 20ms loop time was not achievable using the 8-bit timer, so a 10ms timer was created and the interrupt system was used to keep track of the number of times the timer reached 10ms. This allowed the loop time to be varied easily. The PD control function for this problem is shown below.

Straight Wall PD Controller

```
int pdCalc(int input, int setPoint) {
        double kp = 0.3;
        double kd = 9.0;
        int error = setPoint - input;
        int diff = error - previousError;

        int ret = (int)((kp*error) + (kd*diff));

        previousError = error;

        return ret;
}
```

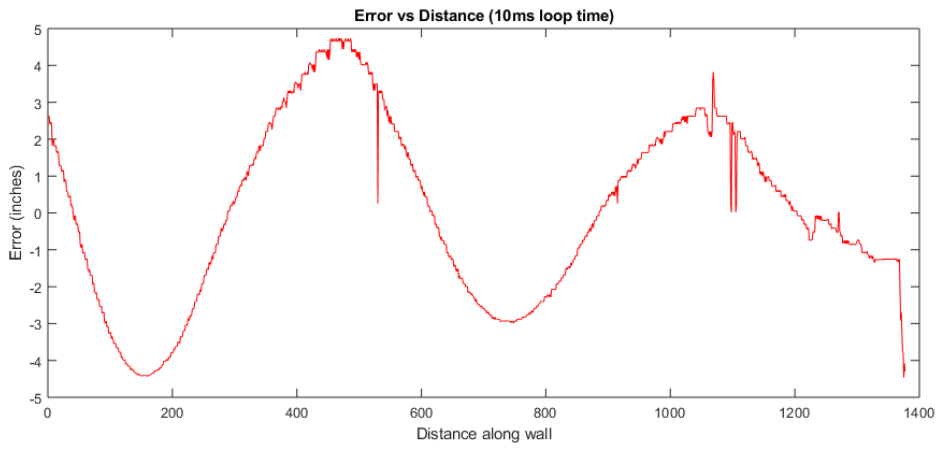
20ms Delay

With a 20ms delay, the robot quickly approaches the wall then quickly levels off around a 0-1" error.



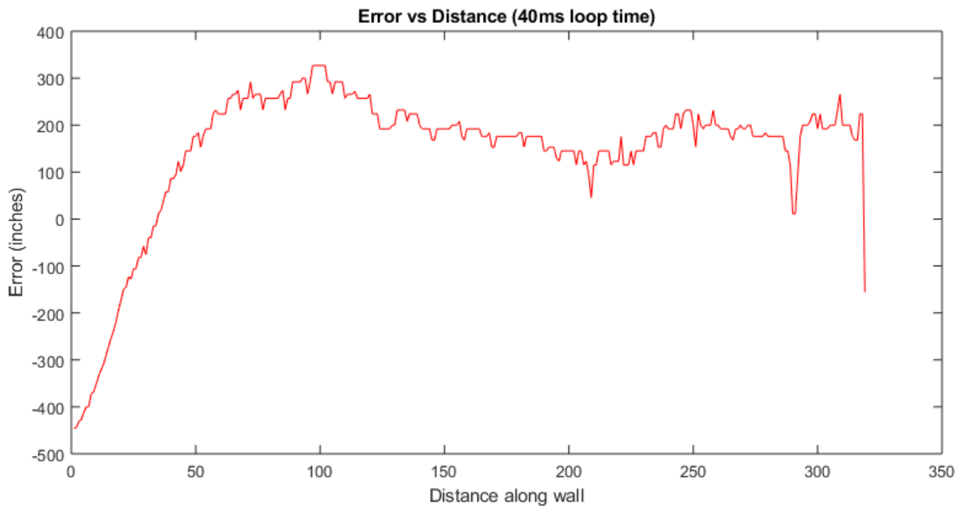
10ms Delay

With a 10ms delay, the robot tends to fluctuate much more, but the error still goes down over time.



40ms Delay

With a 40ms delay, the path of the robot is much smoother.



Circular Wall

I could not find Kp and Kd values that worked for the circular wall, mostly due to time constraints and the fact that it was harder to test this section. An attempted PD controller is shown below.

Circular Wall PD Controller

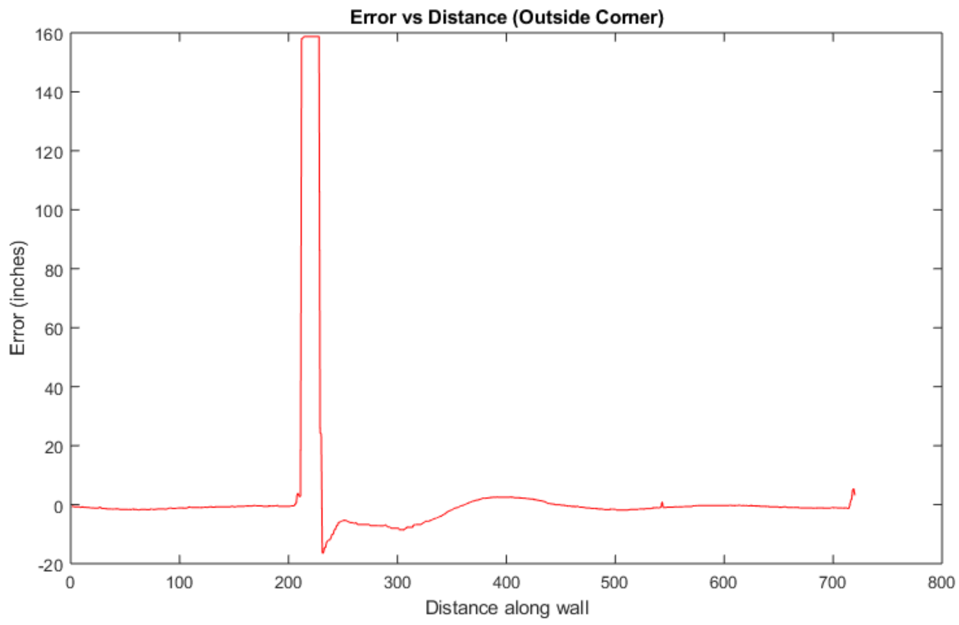
```
int pdCalcCirc(int input, int setPoint) {
        double kp = 10.0;
        double kd = 0.0;

        int error = setPoint - input;
        if (!inited) {
                previousError = error;
                inited = 1;
        }
        if (error < -3000 || error > 300) {
                error = -3000;
        }
        int diff = error - previousError;
        int ret = (int)((kp*error) + (kd*diff));

        previousError = error;

        return ret;
}
```

Outside Corner

For this problem, I used the same Kp and Kd values from the straight wall portion, but when a sudden change in distance was detected, the Kp and Kd values were changed to much more aggressive values (Kp = 2.0, Kd = 0.15). This allowed the robot to turn sharply until it found the wall. I also found that when the sensor could not accurately find the distance to the wall, the value fluctuated wildly and cause significant errors. In the graph you can see where the corner was reached, as the error jumps to ≈160 inches. To prevent this problem, I kept the PD controller input within a specific range. The code for this problem is shown below.

Corner Wall PD Controller

```
int pdCalcSquare(int input, int setPoint) {
        double kp = 0.3;
        double kd = 9.0;

        int error = setPoint - input;
        if (!inited) {
                previousError = error;
                inited = 1;
        }
        if (error < -2500 || error > 2500) {
                error = -2500;
        }
        int diff = error - previousError;


        if (error < 200 && error > -200)
                aligned = 1;

        if (aligned) {
                if (diff <= -300 && !passingDoor) {
                        passingDoor = 1;
                }
                else if (diff >= 300 && passingDoor) {
                        passingDoor = 0;
                }

                if (passingDoor) {
                        kp = 2.0;
                        kd = 0.15;
                }
        }
```

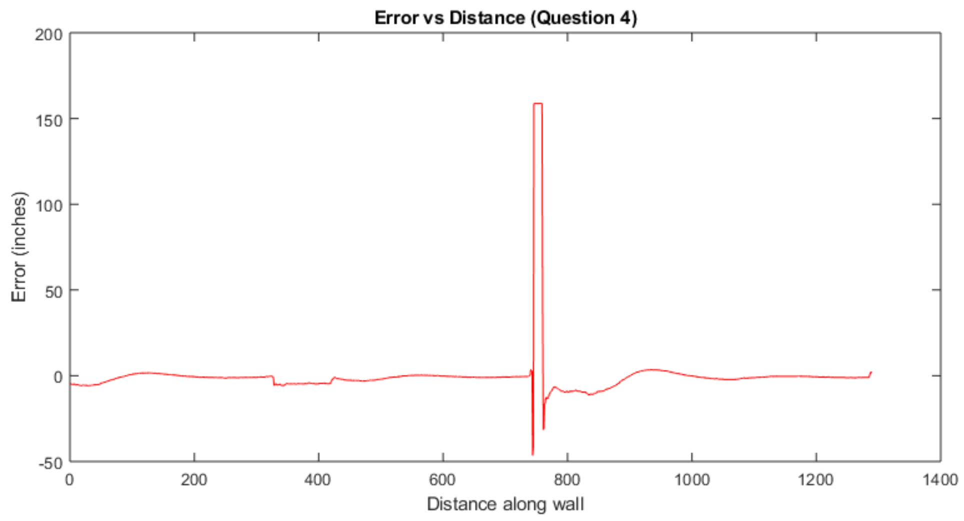


Question 4

For this problem, the code was very similar to the corner wall section of question 3. Instead of changing Kp and Kd a "corner", or sudden change, is detected, we instead increment a counter. When the counter reaches the number of doors we wish to travel past, we turn into the next door using the same aggressive Kp and Kd from the corner section of Q3. For our data, the data did not start collecting soon enough to see the first door, but you can see a sudden & maintained dip around the 300th sample . This is the second door as seen by the sensor. Then, the large error seen around 750 is the open door the robot turns into. The code for this problem is shown below.

Door PD Controller

```
int pdCalcDoor(int input, int setPoint, int numDoors) {
        double kp = 0.3;
        double kd = 9.0;

        int error = setPoint - input;
        if (!inited) {
                previousError = error;
                inited = 1;
        }
        if (error < -2500 || error > 2500) {
                error = -2500;
        }

        int diff = error - previousError;


        if (error < 200 && error > -200)
                aligned = 1;

        if (aligned) {
                if (diff <= -300 && !passingDoor) {
                        doorCount ++;
                        passingDoor = 1;
                }
                else if (diff >= 300 && passingDoor) {
                        passingDoor = 0;
                }

                if (passingDoor && doorCount < numDoors) {
                        previousError = error;
                        return 0;
                }
                else if (passingDoor && doorCount == numDoors) {
                        kp = 2.0;
                        kd = 0.15;
                }
        }

        int ret = (int)((kp*error) + (kd*diff));

        previousError = error;

        return ret;
}
```



Code

File 1: main.c

```
/*
 * mobilebotsFinal.c
 *
 * Created: 5/6/2019 2:09:38 PM
 * Author : colemanct
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <string.h>
#include "pwm.h"
#include "serial.h"
#include "pd.h"
#include "ir.h"
#include "timer.h"

#define setpoint 1800

#define MAX_SPEED 2250
#define DESIRED_SPEED 3000
#define MIN_SPEED 3700

int fitInBounds(int val) {
        val = (val < MAX_SPEED) ? MAX_SPEED : val;
        val = (val > MIN_SPEED) ? MIN_SPEED : val;
        return val;
}

ISR (TIMER0_COMPA_vect) {
        timerInc();
}

int main(void) {
        serialInit();
        pwmInit();
        irInit();
        sei();
        timerInit(20);
        char tempString[64] = {0};

        int rMotorSpeed = DESIRED_SPEED;
        int lMotorSpeed = DESIRED_SPEED;

    while (1) {
                timerReset();

                uint16_t val = irLinRead();

                int pd = pdCalcDoor(val, setpoint, 3);

                lMotorSpeed = DESIRED_SPEED - pd/2;
                rMotorSpeed = DESIRED_SPEED + pd/2;
                lMotorSpeed = fitInBounds(lMotorSpeed);
                rMotorSpeed = fitInBounds(rMotorSpeed);
                pwmSet(LEFT_MOTOR, lMotorSpeed);
                pwmSet(RIGHT_MOTOR, rMotorSpeed);

                sprintf(tempString, "%d\r\n", (setpoint - val));
                serialPrint(tempString, strlen(tempString));

                while(!timerDone()) {}
    }
}
```

File 2: serial.h

```
/*
 * serialFcns.h
 *
 * Created: 3/1/2019 2:33:15 PM
 *  Author: colemanct
 */


#ifndef SERIAL_H_
#define SERIAL_H_

void serialInit();
void serialPrint(char* str, int length);

#endif /* SERIALFCNS_H_ */
```

File 3: serial.c

```
/*
 * serialFcns.c
 *
 * Created: 3/1/2019 2:32:30 PM
 *  Author: colemanct
 */
#include <avr/io.h>
#include <avr/interrupt.h>

void serialInit() {
        UBRR0H = (unsigned char)(103 >> 8);
        UBRR0L = (unsigned char)(103 & 0xFF);

        UCSR0B |= (1 << RXEN0)|(1<<TXEN0);
}

void serialPrint(char* str, int length) {
        for (int i = 0; i < length; i++) {
                while((UCSR0A & (1 << UDRE0)) == 0) {}
                UDR0 = str[i];
        }
}
```

File 4: ir.h

```
/*
 * ir.h
 *
 * Created: 4/3/2019 3:22:30 PM
 *  Author: colemanct
 */


#ifndef IR_H_
#define IR_H_

#define IR_PIN  0

void irInit();
uint16_t irRead();
uint16_t irLinRead();

#endif /* IR_H_ */
```

File 5: ir.c

```
/*
 * ir.c
 *
 * Created: 4/3/2019 3:22:19 PM
 *  Author: colemanct
 */
 #include <avr/io.h>

void irInit() {
        ADMUX |= (1 << REFS0);
        ADCSRA |= (1 << ADPS2)|(1 << ADPS1)|(1 << ADPS0);
        DIDR0 |= (1 << ADC0D);
        ADCSRA |= (1 << ADEN);
}

uint16_t irRead() {
        ADCSRA |= (1 << ADSC);

        while ( (ADCSRA & (1 << ADSC)) );

        return ADC;
}

uint16_t irLinRead() {
        uint16_t val = irRead();

        return (500000/val);
}
```

File 6: pwm.h

```
/*
 * pwm.h
 *
 * Created: 4/3/2019 3:18:17 PM
 *  Author: colemanct
 */


#ifndef PWM_H_
#define PWM_H_

#define RIGHT_MOTOR 1
#define LEFT_MOTOR 2

void pwmInit();
void pwmSet(int motor, int usHigh);

#endif /* PWM_H_ */
```

File 7: pwm.c

```
/*
 * pwm.c
 *
 * Created: 3/6/2019 2:52:12 PM
 *  Author: colemanct
 */
#include <avr/io.h>

#define TOP 0x9C40

void pwmInit() {
        DDRB |= (1 << DDB1);
        DDRB |= (1 << DDB2);
        OCR1A = 0;
        OCR1B = 0;
        ICR1 = TOP;

    TCCR1A |= (1 << COM1A1)|(1 << COM1B1);
    // set none-inverting mode

    TCCR1A |= (1 << WGM11);
    TCCR1B |= (1 << WGM12)|(1 << WGM13);
    // set Fast PWM mode using ICR1 as TOP

    TCCR1B |= (1 << CS11);
    // START the timer with no prescaler
}

void pwmSet(int motor, int usHigh) {
        int val = usHigh / 5;
        val *= 4;
        if (motor == 1)
                OCR1A = (int) val;
        else if (motor == 2)
                OCR1B = (int) val;
}
```

File 8: pd.h

```c
/*
 * pd.h
 *
 * Created: 4/6/2019 5:19:47 PM
 *  Author: colemanct
 */


#ifndef PD_H_
#define PD_H_

int pdCalc(int input, int setPoint);
int pdCalcCirc(int input, int setPoint);
int pdCalcSquare(int input, int setPoint);
int pdCalcDoor(int input, int setPoint, int numDoors);

#endif /* PD_H_ */
```

File 9: pd.c

```c
/*
 * pd.c
 *
 * Created: 4/6/2019 5:19:34 PM
 *  Author: colemanct
 */

#include <avr/io.h>


int previousError = 0;
int inited = 0;
int aligned = 0;
int doorCount = 0;
int turningCorner = 0;
int passingDoor = 0;

int pdCalc(int input, int setPoint) {
        double kp = 0.3;
        double kd = 9.0;
        int error = setPoint - input;
        int diff = error - previousError;

        int ret = (int)((kp*error) + (kd*diff));

        previousError = error;

        return ret;
}

int pdCalcCirc(int input, int setPoint) {
        double kp = 10.0;
        double kd = 0.0;

        int error = setPoint - input;
        if (!inited) {
                previousError = error;
                inited = 1;
        }
        if (error < -3000 || error > 300) {
                error = -3000;
        }
        int diff = error - previousError;

        int ret = (int)((kp*error) + (kd*diff));

        previousError = error;

        return ret;
}

int pdCalcSquare(int input, int setPoint) {
        double kp = 0.3;
        double kd = 9.0;

        int error = setPoint - input;
        if (!inited) {
                previousError = error;
                inited = 1;
        }
        if (error < -2500 || error > 2500) {
                error = -2500;
        }
        int diff = error - previousError;


        if (error < 200 && error > -200)
                aligned = 1;

        if (aligned) {
                if (diff <= -300 && !passingDoor) {
                        passingDoor = 1;
                }
                else if (diff >= 300 && passingDoor) {
                        passingDoor = 0;
                }

                if (passingDoor) {
                        kp = 2.0;
                        kd = 0.15;
                }
        }

        int ret = (int)((kp*error) + (kd*diff));

        previousError = error;

        return ret;
}

int pdCalcDoor(int input, int setPoint, int numDoors) {
        double kp = 0.3;
        double kd = 9.0;

        int error = setPoint - input;
        if (!inited) {
                previousError = error;
                inited = 1;
        }
        if (error < -2500 || error > 2500) {
                error = -2500;
        }

        int diff = error - previousError;


        if (error < 200 && error > -200)
                aligned = 1;

        if (aligned) {
                if (diff <= -300 && !passingDoor) {
                        doorCount ++;
                        passingDoor = 1;
                }
                else if (diff >= 300 && passingDoor) {
                        passingDoor = 0;
                }

                if (passingDoor && doorCount < numDoors) {
                        previousError = error;
                        return 0;
                }
                else if (passingDoor && doorCount == numDoors) {
                        kp = 2.0;
                        kd = 0.15;
                }
        }

        int ret = (int)((kp*error) + (kd*diff));

        previousError = error;

        return ret;
}
```

File 10: timer.h

```
/*
 * timer.h
 *
 * Created: 5/12/2019 11:43:57 AM
 *  Author: colemanct
 */


#ifndef TIMER_H_
#define TIMER_H_

int timerCount;

void timerInit();
void timerInc();
int timerDone();
void timerReset();

#endif /* TIMER_H_ */
```

File 11: timer.c

```c
/*
 * timer.c
 *
 * Created: 5/12/2019 11:43:44 AM
 *  Author: colemanct
 */

#include <avr/io.h>
#include "serial.h"

int timerCount = 0;
int comp = 0;

void timerInit(int msec) {

    // Set the Timer Mode to CTC
    TCCR0A |= (1 << WGM01);

    // Set the value that you want to count to
    OCR0A = 0x9C;

        comp = msec/10;

    TIMSK0 |= (1 << OCIE0A);    //Set the ISR COMPA vect

    TCCR0B |= (1 << CS02) | (1 << CS00);
    // set prescaler to 256 and start the timer
}

void timerInc() {
        timerCount ++;
}

int timerDone() {
        return (timerCount >= comp);
}

void timerReset() {
        timerCount = 0;
}
```