RENSSELAER POLYTECHNIC INSTITUTE

Laboratory 4 Report

Yilu Zhou, Z Jin, Zhongqi Liu

Section 02, Side 9B, 10B, 11B

supervised by

Dr. Jeffrey BRAUNSTEIN

graded by

Syed NAQVI

June 21, 2019

1 INTRODUCTION

1.1 PURPOSE

The purpose of lab 4 is

1. to integrate the compass and ranger systems developed in the previous lab to control the steering and driving. The terminal emulator (SecureCRT or PuTTY) through the wireless serial link will be used to collect data, set some options in the program and set the speed.

2. to use Analog to Digital Conversion as was implemented in Lab 2 to read an A/D input

3. to display control information on LCD display screen or the terminal and enter variables using either the keypad or the terminal keyboard.

1.2 OVERVIEW OF STEERING & DRIVE CONTROL

In the $C8051F020$ micro-controller, the control of the steering servo and drive motor is achieved by the use of different pulse width modulation (PWM). In order to control the motor correctly, the period of the PWM is set to 20 ms. Instead of using the $Timer0$ in Lab 1 and 2, the students need to use the $PCA0$ to count the frequency of the chip on the micro-controller. However, if the students use the default setting of 16-bit counter and $SYSCLK/12$ mode on the $PCA0$, they will not get the 20 ms period. Instead, they will get the 35.5 ms period. Therefore, to achieve the 20 ms period, they need to set the PCA_START to a higher level instead of 0 to get a 20 ms period. In this lab, we use $PCA\_START = 28614$. This value is calculated by using the equation:

$$\frac{20\,ms}{35.5\,ms} = \frac{65535 - PCA\_START}{65535}$$

$$\boxed{PCA\_START = 28614.}$$

After referring to the lab manual, the pulse width for the drive motor to do full forward is about 1.9 ms and full reverse is about 1.1ms. For the steering servo the pulse width for the steering servo to do hard left is about 0.9 ms and hard right is about 2.1 ms. After calculation, we write the following code:

```
#define DRIVE_MOTOR_NEUT 2765

#define STEERING_SERVO_NEUT 2730

unsigned int DRIVE_MOTOR_MAX = 3505;

unsigned int DRIVE_MOTOR_MIN = 2028;

unsigned int STEERING_SERVO_LEFT = 2230;

unsigned int STEERING_SERVO_RIGHT = 3230;
```

The above codes set the maximum and minimum pulse width for the drive motor and steering servo. If the pulse width is beyond that range, the drive motor and steering servo will stop working. If the students want to control the pulse width modulation for the drive motor and steering servo, they need to set the high and low byte of the pulse width separated as the high byte of the pulse width will be overflowed if the students forget to do so. Therefore, the functions to control the pulse width are the following:

```
void Drive_Motor(void)

{

    // This function will control the drive motor.

    PCA0CPL2 = 0xFFFF - DRIVE_MOTOR_PW;

    // Set the low byte of CEX2.

    PCA0CPH2 = (0xFFFF - DRIVE_MOTOR_PW) >> 8;

    // Set the high byte of CEX2.

}


void Steering_Servo(void)

{

    // This function will control the steering servo.

    PCA0CPL0 = 0xFFFF - STEERING_SERVO_PW;

    // Set the low byte of CEX0.

    PCA0CPH0 = (0xFFFF - STEERING_SERVO_PW) >> 8;

    //Set the high byte of CEX0.

}
```

Note that the data register is determined by the wiring and the use of $PCA0$ ports. In this lab, we set the $PCA0$ to

```
XBR0 = 0x27; // Configure crossbar with UART, SPI, SMBus, and CEX
   channels.
```

This means that $CEX0$ is configured for the steering servo and $CEX2$ is configured for the drive motor. Therefore, these ports' pulse width modulation is determined by the above codes. These functions will be called from the main function to change the pulse width modulation for the drive motor and steering servo.

According to the laboratory guide, the orientation of the steering servo depends on the data from the compass and the rotation of the drive motor depends on the data from the ultrasonic ranger. Therefore, separate functions to read the data from the ranger and the compass is needed. There are some limitations of the ultrasonic ranger needed to be considered:

1. Some time is needed to read the ultrasonic echo.
2. The data is needed to be read in cm instead of in.

Therefore, the students need to first write data to the registers of the ultrasonic ranger and then read the data from the ranger. The command to write to the register is includes in the $i2c.h$ header file. The argument is

```
void Ping_Ranger(void)
{
    // This function ping the ranger for later data read.
    RANGER_DATA[0] = 0x51;                             // Read the
       data in cm
    i2c_write_data(RANGER_ADDRESS, 0, RANGER_DATA, 1); // write one
       byte of data to reg 0 at addr, read in cm.
}
```

Note that the function $i2c\_write\_data$ takes an array instead of a single variable for the data. The above function write the data `0x51` to the register 0 of the ultrasonic ranger. The $RANGER\_ADDRESS$ is defined as global variable as `0xE0` and used in the function as well. Each time when the students want to read the data from the ultrasonic ranger, they need to first ping the ranger in order to read the data in cm.

When considering the compass, the steering servo needs the data from the compass to do the adjustment continuously. Therefore, the function $Read\_Compass$ is called every 20ms in order to coincide with the period of the PWM. Because the radius of turn is considered in this lab, the $HEADING\_ERROR$ is designed to be in the range of -1800 to 1800. The $HEADING\_ERROR$ is calculated as the following:

$$\text{HEADING\_ERROR} = \text{HEADING} - \text{TARGET\_HEADING}.$$

When the value is out of ranger, the codes need to subtract or add 3600 to offset the differences in the value. By doing so, we make sure that the car will not make any turn larger than ± 1800. After the $HEADING\_ERROR$ is calculated, we need to change the $STEERING\_SERVO\_PW$ to make the servo change to the target direction.

2 RESULTS, PLOTS, ANALYSIS OF PLOTS, & CONCLUSIONS

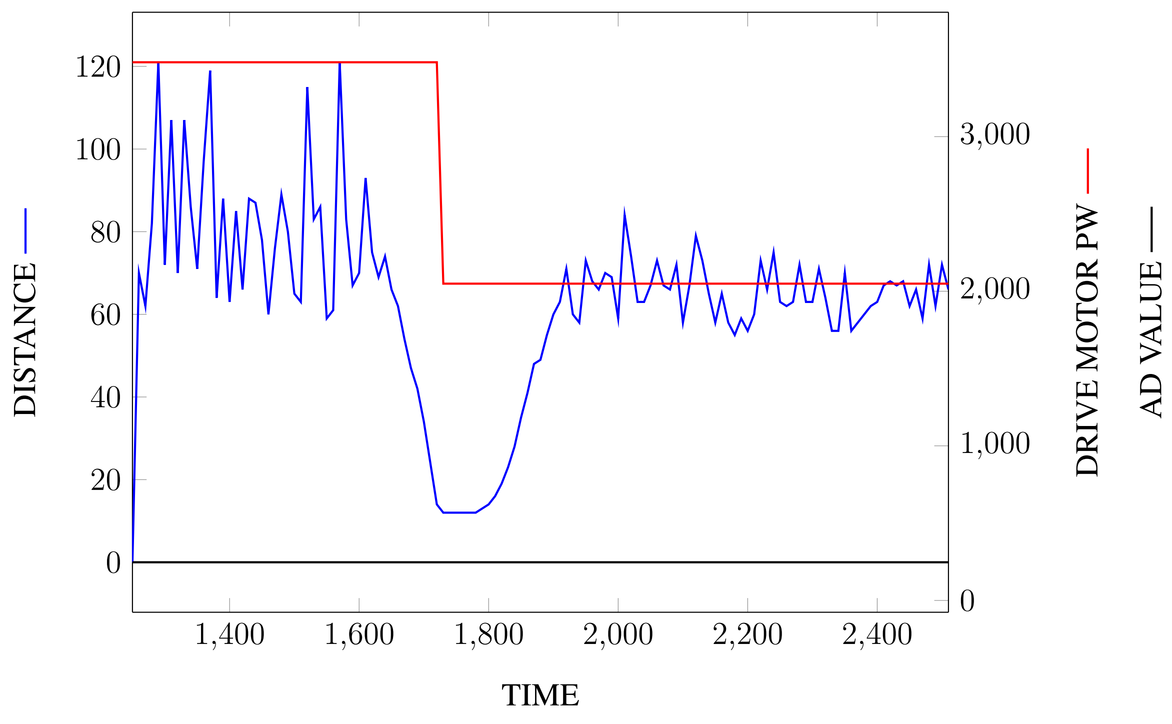
2.1 RESULTS

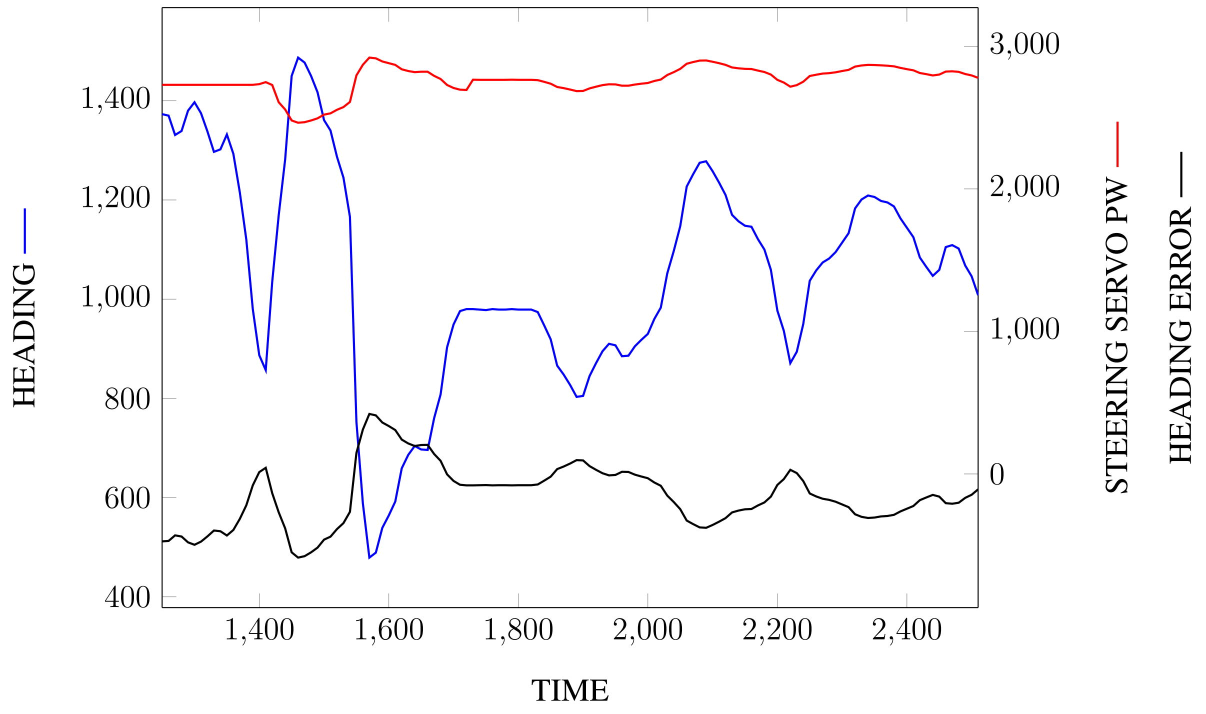
The overall results of the lab is checked by the TA. The car can return to the departure place after it returns from the far end of the paper. The data is collected and plotted in the next subsection.

2.2 PLOTS

Several runs of the codes is recorded. The data is gathered by the function $Flight\_Recorder$ and the data is in $.csv$ format. As three different variables in needed to be included in one plot, The combo graph is used with different vertical axis unit on one single graph. All the unit of horizontal axis of the plots in this subsection is $\times 20\ ms$ as it is the same as the $PCA\_COUNTER$ which counts the $PCA0$ overflows.

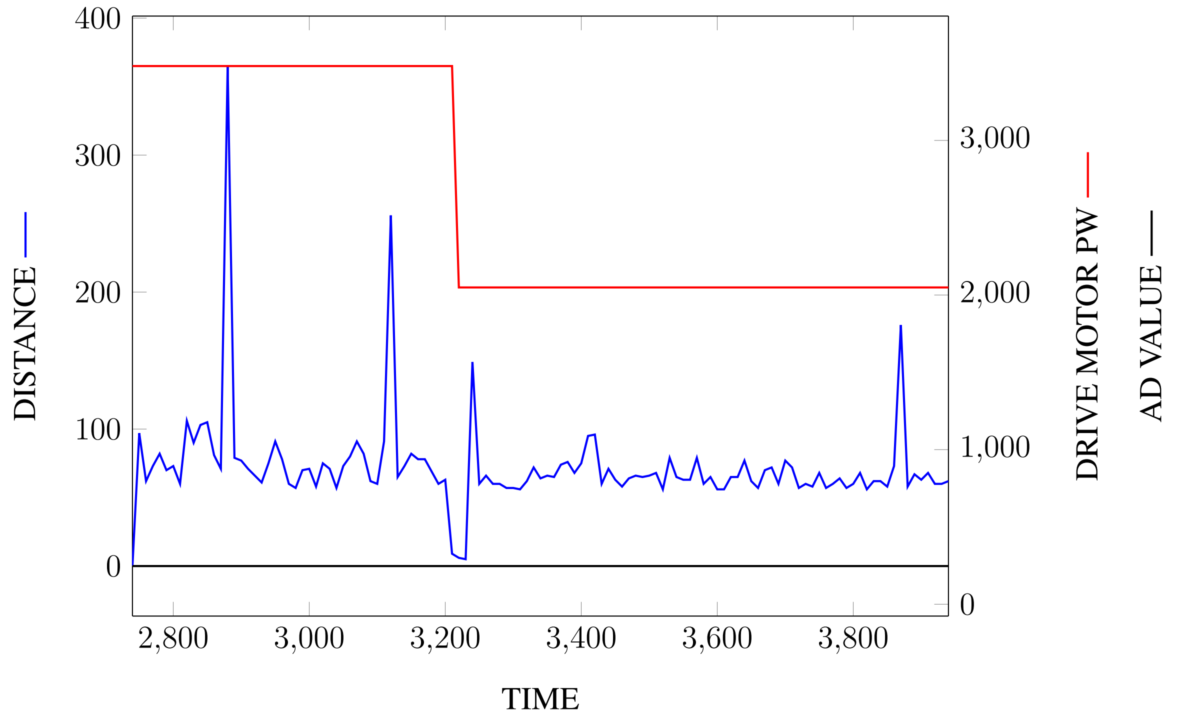
2.2.1 HEADING = 900, STEERING_SERVO_GAIN = 500 & DRIVE_MOTOR_GAIN = 740

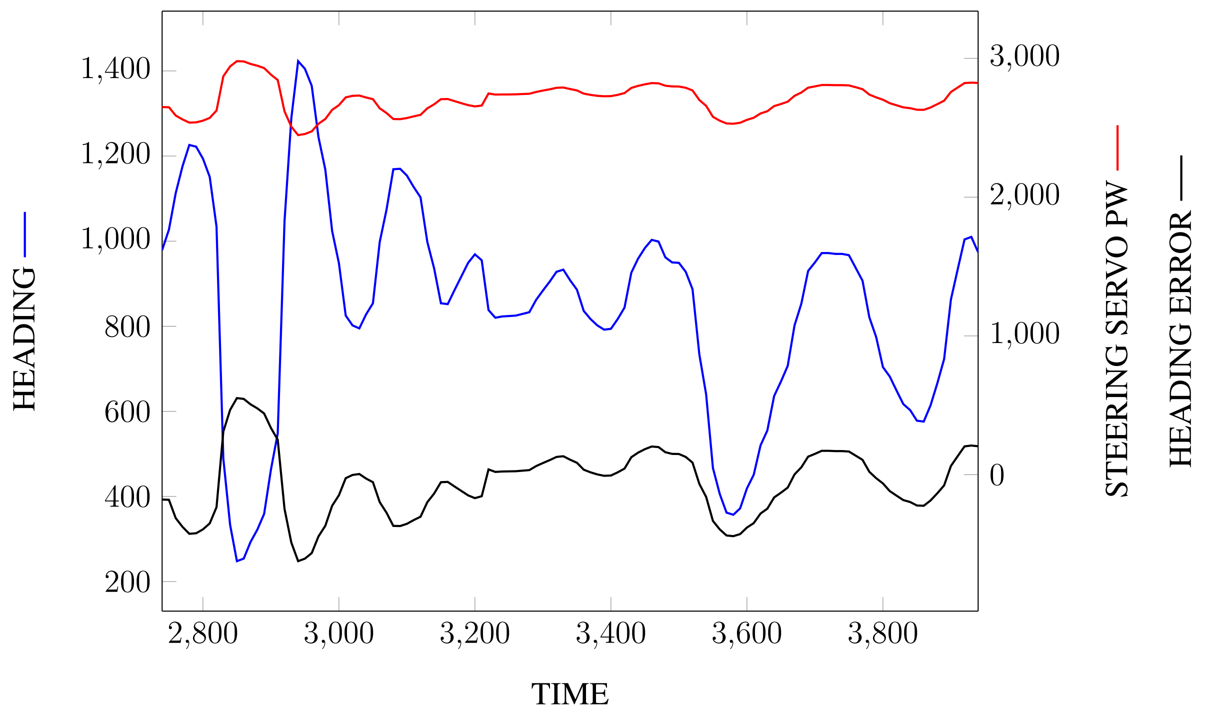




When the $DISTANCE$ is less than 20 cm happens at 1700×20 ms,the motor changes from forward to reverse. When the $HEADING$ decreases, the $STEERING\_SERVO\_PW$ increases. When the $HEADING$ increases, the $STEERING\_SERVO\_PW$ decreases because the $HEADING\_ERROR$ is reversed.

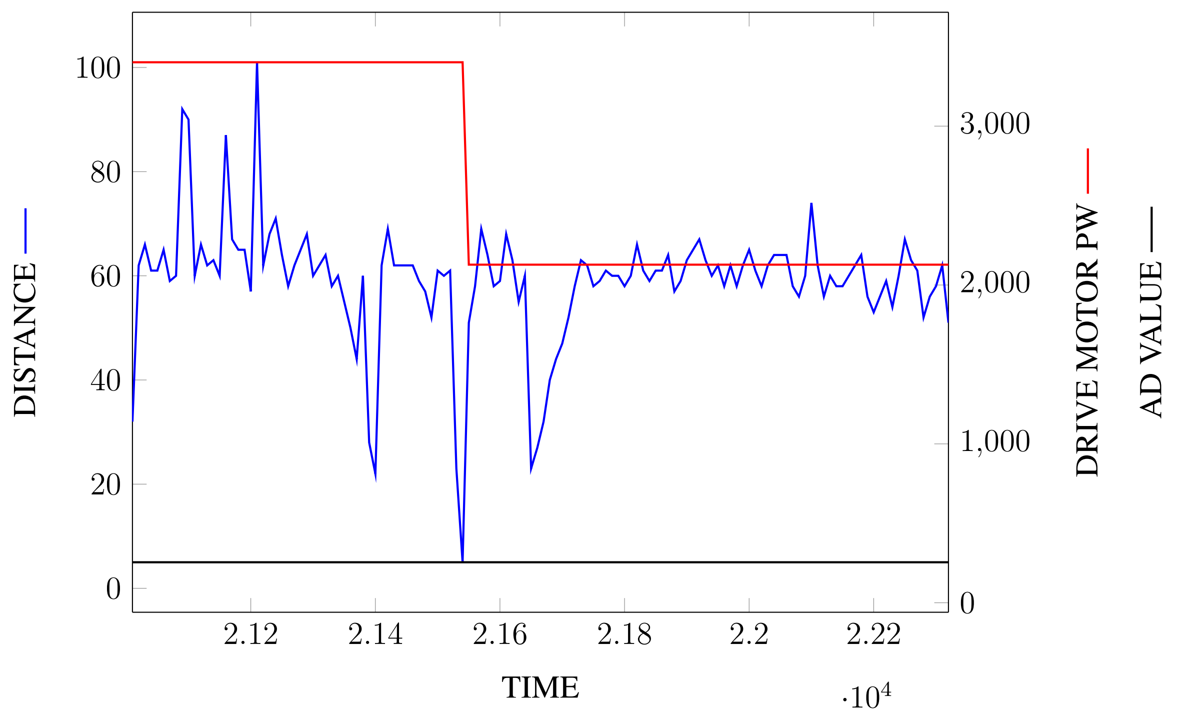
2.2.2 HEADING = 800, STEERING_SERVO_GAIN = 500 & DRIVE_MOTOR_GAIN = 740

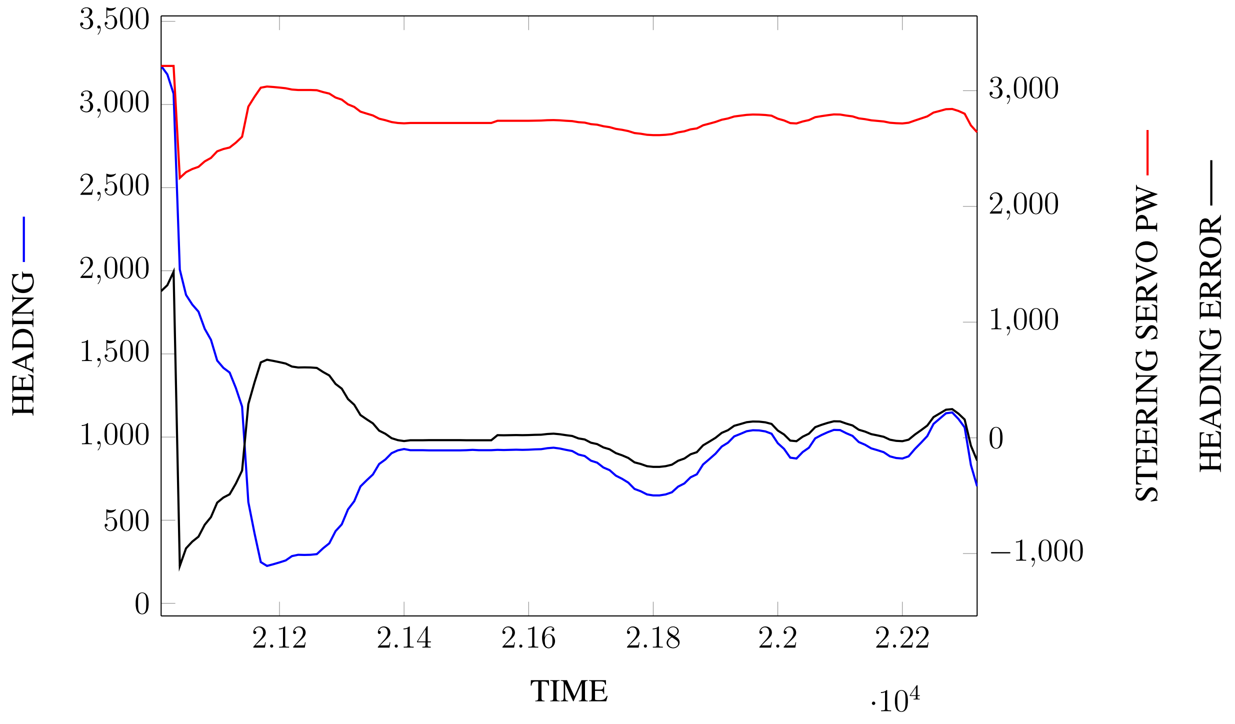




When the drive motor going forward, the $HEADING$ and $HEADING\_ERROR$ has inverse relationship. When the drive motor going reverse, they have linear relationship.

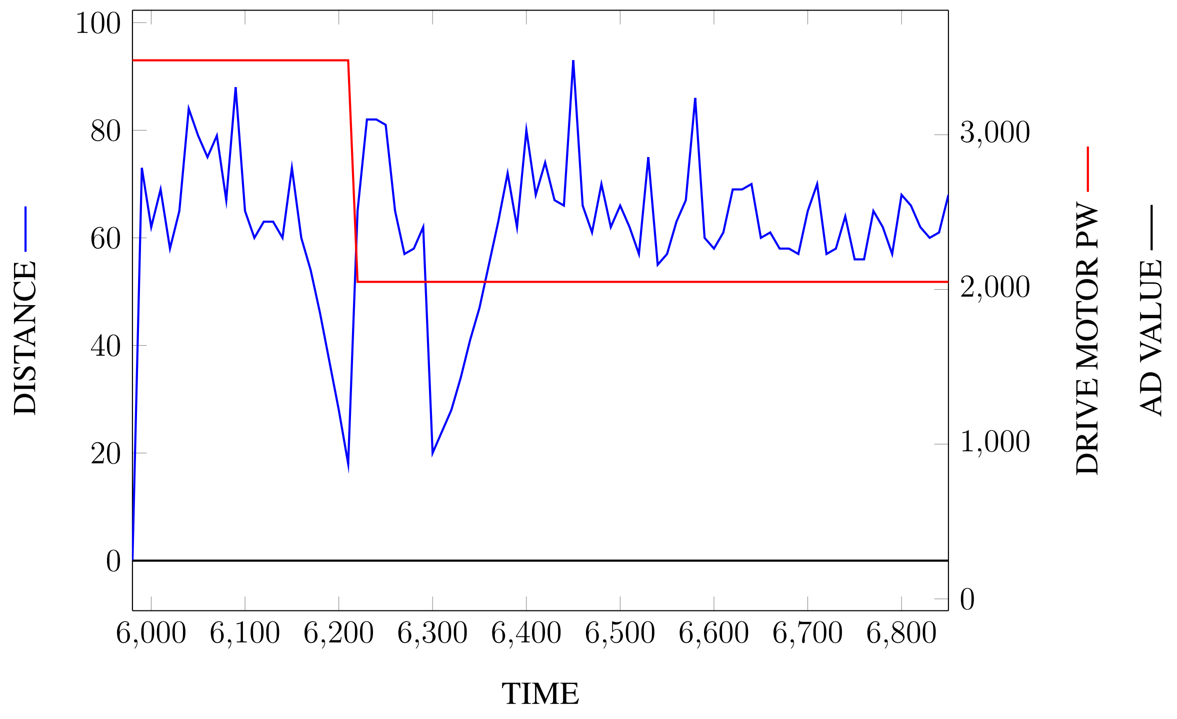
2.2.3 HEADING = 900, STEERING_SERVO_GAIN = 500 & DRIVE_MOTOR_GAIN = 640

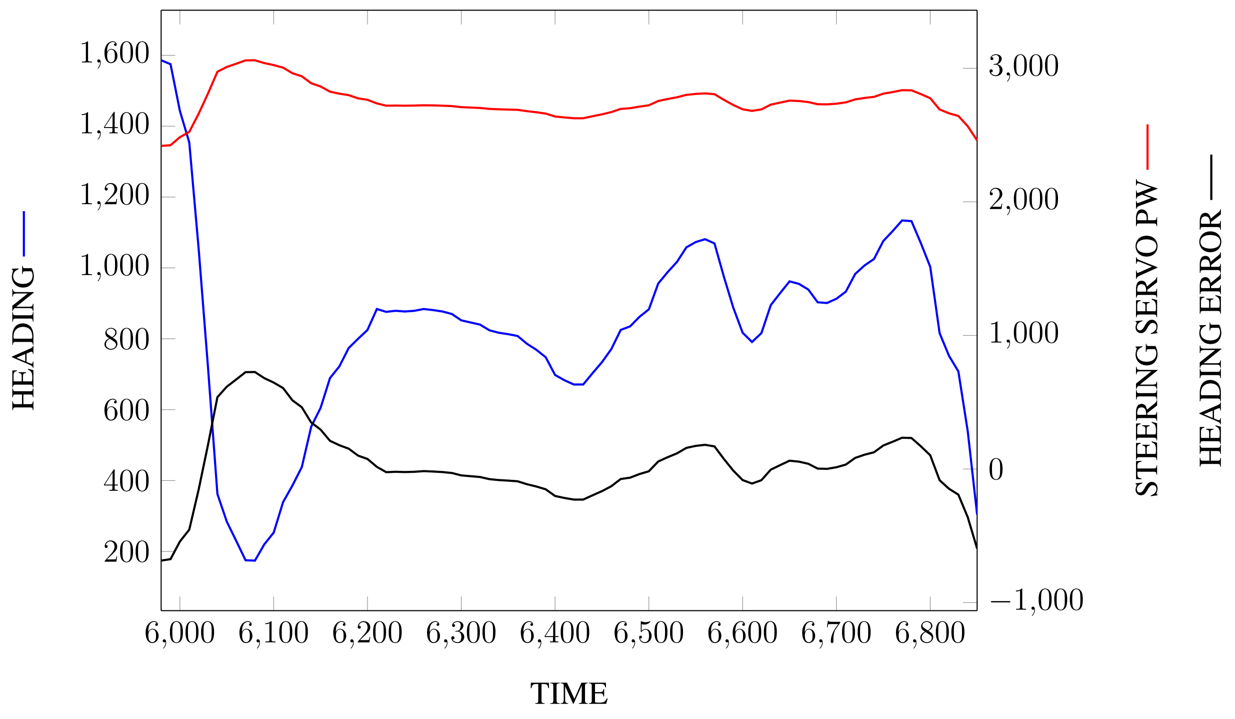




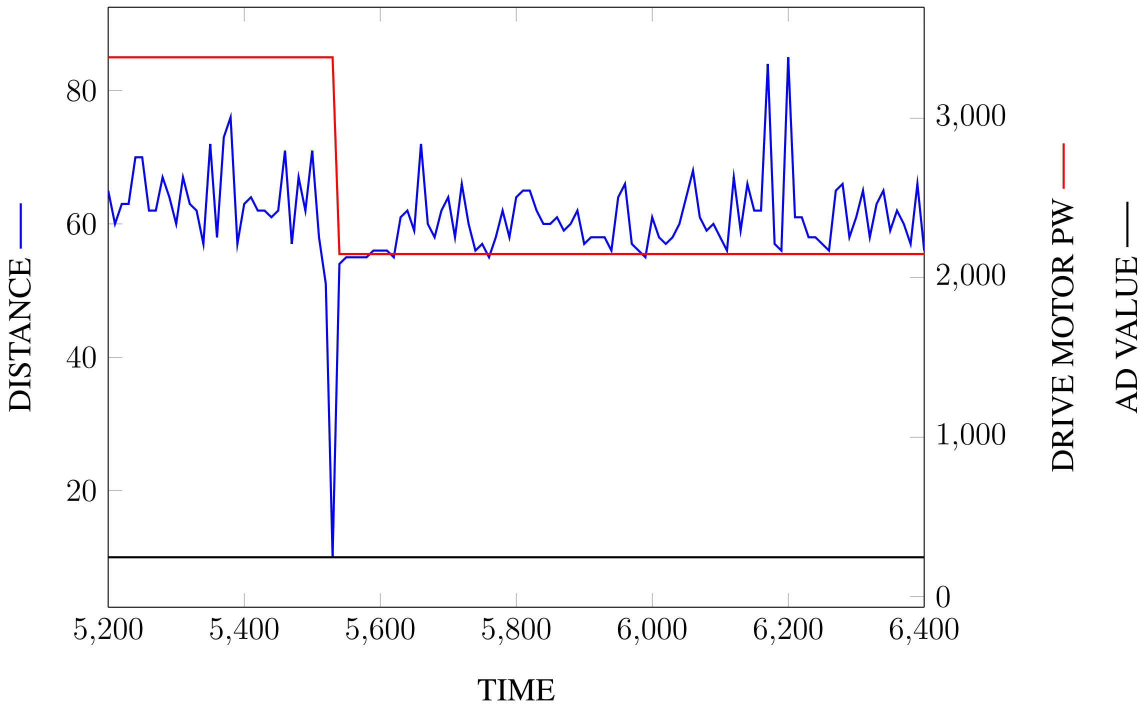
There are several points when the $DISTANCE$ is close to 20 cm at time 2140×20 ms and 2170×20 ms, but the drive motor only changes when the $DISTANCE$ is below 20 cm.

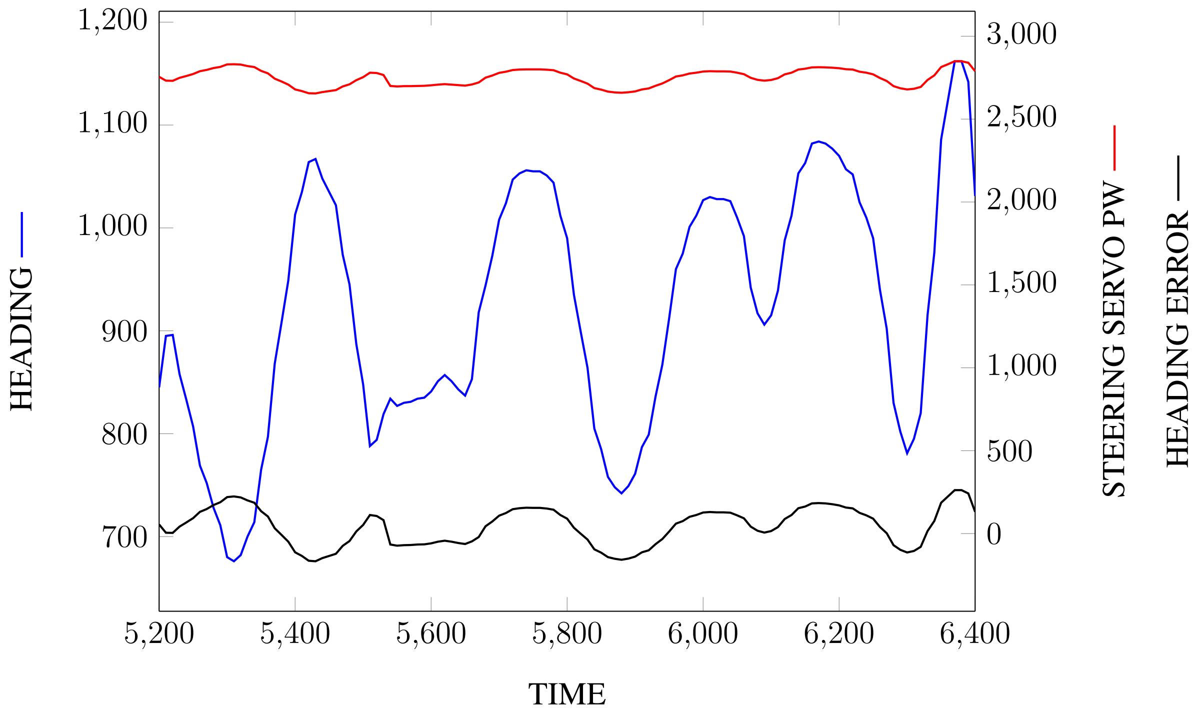
2.2.4 HEADING = 900, STEERING_SERVO_GAIN = 400 & DRIVE_MOTOR_GAIN = 740





2.2.5 HEADING = 900, STEERING_SERVO_GAIN = 400 & DRIVE_MOTOR_GAIN = 640





2.3 VERIFICATIONS

In the lab guide, there is specific info about the flow of the program.

In general, the user should first input the $TARGET\_HEADING$, $STEERING\_SERVO\_GAIN$ and $DRIVE\_MOTOR\_GAIN$ from the keypad and the entered values should be shown on the LCD screen. Then, the micro-controller should check the status of the slide switch. If the slide switch is on, the car should be going and the data should be sent through Putty. If the slide switch is off, the car should wait for the slide switch to be turned on by the user. The car should be going for at least 3 seconds before reading the data from the ranger and change the rotation of the drive motor. However, the compass should be read from time to time to make sure that the target heading is aligned with the front of the car by changing the pulse width modulation of the steering servo. Once there are some objects within 20 cm, the drive motor should go reverse. Ideally, the car will return to the place where it departs.

In order to test the final results, we ask one student to stand where the car departs. Then another student will set the target heading and gains using the keypad and release the car from where the first student stands. The third student will trying to chasing the car and let the drive motor turn reverse by blocking the ultrasonic ranger using a book. The car is expected to return to the first student if the code and wiring is correct.

To ensure that the car will not reverse the drive motor until 3 seconds has passed, we ask one student to stand in front of the car. After the data is set and slide switch is on, the car did not reverse the drive motor at first. It reverses the drive motor after 3 seconds has passed. Then, after another 3 seconds, the car turns the drive motor again. Therefore, this specification is satisfied.

The wiring of the compass is crucial to the success of the whole lab as it is responsible for controlling the steering servo. Before using the compass, one needs to calibrate it first. According to the guide $Compass\_calibration.pdf$, to calibrate the compass, SMB, SCL and SDA lines should be disconnected from the compass during the procedure. However, the $+5V$ and the GND should be connected to the compass as well. After the wiring is checked, follow the procedure:

1. Set the compass module flat, pointing North. Press and release the switch,

2. Set the compass module flat, pointing East. Press and release the switch,

3. Set the compass module flat, pointing South. Press and release the switch,
4. Set the compass module flat, pointing West. Press and release the switch.

If the data read is still not correct after one calibration, repeat the above procedure until the result is satisfactory.

2.4 PROBLEMS ENCOUNTERED

1. At first, there are some mistakes made after reading the lab guide. Someone thought that the car should continue going forward if there is no objects within 20 cm after the car turned reverse.
2. When the codes is checked and we try to test the car on the ground, after the ranger detected an object within 20cm, instead of turning back directly, it tries to take a U-turn and then goes back.
3. The final problem is that when we trying to connect the RF Transceiver Module to the car, no matter what we do, the PuTTy will not show any information from the car.

2.5 SOLUTIONS

1. The logic in the codes is solved by simply rearranged the codes as the whole codes is modular.
2. This problem is caused by a bug in the codes. When the ultrasonic ranger detects an object, it will tries to change the steering servo, which should not be changed, at that point. After deleting the codes controlling the steering servo, the car is capable of returning to the departure points.
3. After asking professor for help, he concludes that the wiring is correct and says that we can only use cable to connect to the car from the laptop. This problem is only partially solved therefore.

2.6 CONCLUSIONS

The results of the Lab 4 is checked by the TA and the trace of the car is stable after several tests.

2.7 SUGGESTED IMPROVEMENTS TO HW AND SW

The HW is really helpful for the Lab 4. The homework has already help the students combine the codes from steering servo and ultrasonic ranger part of Lab 3. Therefore, the codes of Lab 4 only needs minor changes to the ordinary codes. Also, the majority of the wiring of the EVB is completed after finishing Lab 3 and no big changes is needed to complete Lab 4.

However, it would be better to combine the Lab 4 with Lab 3 as the majority of the codes and wiring are the same. This is also helpful for the Lab 5 and Lab 6 as more time is available for later labs.

3 CODE

```
#include <stdio.h>

#include <stdlib.h>

#include <c8051_SDCC.h>

#include <i2c.h>


// Define constant

#define PCA_START 28614          // The start value for the 20ms
   period.

#define DRIVE_MOTOR_NEUT 2765    // The PW for the drive motor to stop
   .

#define STEERING_SERVO_NEUT 2730 // The PW for the steering servo to
   be neutral.


// Global variables

unsigned int DISTANCE = 0;                        // The distance in cm
   read by the ultrasonic sensor.

unsigned int HEADING = 0;                         // The value for
   returning degrees from the compass.

const unsigned char RANGER_ADDRESS = 0xE0;        // The address of the
   ultrasonic ranger.

const unsigned char COMPASS_ADDRESS = 0xC0;       // The address of the
   compass.

const unsigned char POT_ADDRESS = 4;              // The address of POT.

unsigned char RANGER_DATA[2];                     // The variable to
   store the raw data from ranger.

unsigned char COMPASS_DATA[2];                    // The variable to
   store the raw data from the compass.
```

```
unsigned int DRIVE_MOTOR_PW = 0;                // The current PW for
   the drive motor.
unsigned int STEERING_SERVO_PW = 0;             // The current PW for
   the steering servo.
unsigned int PCA_COUNTER = 0;                   // The variable to
   count the PCA overflows.
unsigned char AD_VALUE = 0;                     // The data from POT.
unsigned int DRIVE_MOTOR_MAX = 3505;            // The PW for the drive
    motor to do full forward.
unsigned int DRIVE_MOTOR_MIN = 2028;            // The PW for the drive
    motor to do full reverse.
unsigned int STEERING_SERVO_LEFT = 2230;        // The PW for the
   steering servo to be left.
unsigned int STEERING_SERVO_RIGHT = 3230;       // The PW for the
   steering servo to be right.
unsigned char OUTPUT_MODE = 1;                  // The format of the
   output info.
unsigned int TARGET_HEADING = 0;                // The target heading
   for the car.
unsigned int TARGET_DRIVE_MOTOR_GAIN = 740;     // The target gain of
   the drive motor.
unsigned int TARGET_STEERING_SERVO_GAIN = 500; // The target gain of
   the steering servo.
int HEADING_ERROR = 0;                          // The error in the
   heading.
unsigned int TIME = 0;                          // The counter to count
    the time.
unsigned int RANGER_READ_COUNTER = 150;         // The flag to set the
```

```
   ranger to be read after 3s.

// Component address
__sbit __at 0x94 POT;     // The address of the potentiometer.
__sbit __at 0xB6 SS;      // The address of the slide switch.
__sbit __at 0xDF CF;      // The address of ultrasonic ranger overflow
   flag.
__sbit __at 0xB0 BILED0; // The address of BILED0.
__sbit __at 0xB1 BILED1; // The address of BILED1.

// C8051 initialization funtion.
void Port_Init(void);
void PCA_Init(void);
void XBR0_Init(void);
void Interrupt_Init(void);
void SMB_Init(void);
void ADC_Init(void);
void PCA_ISR(void) __interrupt 9;

// Ultrasonic Ranger and Drive Motor functions
void Drive_Motor(void);
void Speed_Controller(void);
void Ping_Ranger(void);
void Read_Ranger(void);
void Drive_Motor_Init(void);

// Compass and Steering Servo functions
void Steering_Servo(void);
void Direction_Controller(void);
void Read_Compass(void);
void Steering_Servo_Init(void);

// POT functions
void Read_AD_Input(void);
void POT_Reader(void);

// Keypad functions
void Read_Keypad(void);

// UI and data record functions
void Flight_Recorder(void);

// Other functions
unsigned int Is_SS_On(void);
void Turn_BILED_Green(void);
void Turn_BILED_Red(void);
void Turn_BILED_Off(void);
void Reset_PCA_Counter(void);
void Wait_For_1s(void);
void Troubleshooter(void);

void main(void)
{
    // The initialization part of the program.
    Sys_Init();
    putchar('␣');
```

```
Port_Init();
PCA_Init();
XBR0_Init();
Interrupt_Init();
SMB_Init();
ADC_Init();

Turn_BILED_Off();
printf("Embedded␣Control␣Flight␣Recorder\r\n");
Drive_Motor_Init();
Steering_Servo_Init();
//Troubleshooter(); // Should be deleted after the circuit testing
    is completed.
while (1)
{
    if (!SS)
    {
        // When the slide switch is On.
        Reset_PCA_Counter(); // Reset the PCA_COUNTER to 0.
        Turn_BILED_Green();
        DRIVE_MOTOR_PW = DRIVE_MOTOR_MAX; // Set the drive motor
           PW to forward.
        //record data
        Drive_Motor(); // drive the drive motor.

        RANGER_READ_COUNTER = 0; // Reset the ranger read flag.
        while (!SS)
        {
```

```
        Direction_Controller();
        Steering_Servo();
        Drive_Motor();
        if (PCA_COUNTER > 150 || RANGER_READ_COUNTER > 150)
        {
            Speed_Controller();
            Reset_PCA_Counter();
        }
    }
    DRIVE_MOTOR_PW = DRIVE_MOTOR_NEUT;
    STEERING_SERVO_PW = STEERING_SERVO_NEUT;
    Drive_Motor();
    Steering_Servo();
}
else
{
    // When the slide switch is Off, use the keypad to read
       the desired gain and heading
    Read_Keypad(); // Read the input from the keypad
    POT_Reader();
    DRIVE_MOTOR_PW = DRIVE_MOTOR_NEUT;
    Drive_Motor();
    STEERING_SERVO_PW = STEERING_SERVO_NEUT;
    Steering_Servo();
    Reset_PCA_Counter();
    while (SS)
        ;
    lcd_clear();
```

```
            lcd_print("SS␣is␣on.␣Execute␣the␣code.");
            // Set drive motor and steering servo to neutral position.
            RANGER_READ_COUNTER = 0;
        }
    }
}

void Drive_Motor(void)
{
    // This function will drive the DRIVE_MOTOR.
    PCA0CPL2 = 0xFFFF - DRIVE_MOTOR_PW;        // Set the low byte of
       CEX2.
    PCA0CPH2 = (0xFFFF - DRIVE_MOTOR_PW) >> 8; // Set the high byte of
        CEX2.
}

void Speed_Controller(void)
{
    // This function will assign the correct DRIVE_MOTOR_PW according
       to the DISTANCE.
    if (DISTANCE < 20)
    {
        RANGER_READ_COUNTER = 0; // hold the ranger read flag to 0 to
           stop speed change.
        // If the distance is smaller than 20cm, change the direction
           of the drive motor.
        if (DRIVE_MOTOR_PW > DRIVE_MOTOR_NEUT)
        {
            DRIVE_MOTOR_PW = DRIVE_MOTOR_MIN;

            Turn_BILED_Green();

            //TARGET_HEADING += 1800;

        }

        else

        {

            DRIVE_MOTOR_PW = DRIVE_MOTOR_MAX;

            Turn_BILED_Red();

        }

    }

}


void Ping_Ranger(void)

{

    // This function ping the ranger for later data read.

    RANGER_DATA[0] = 0x51;                           // Read the

       data in cm

    i2c_write_data(RANGER_ADDRESS, 0, RANGER_DATA, 1); // write one

       byte of data to reg 0 at addr, read in cm.

}


void Read_Ranger(void)

{

    // This function read the distance from the ranger.

    i2c_read_data(RANGER_ADDRESS, 2, RANGER_DATA, 2);

       // Read the data from the ultrasonic ranger.

    DISTANCE = ((unsigned int)RANGER_DATA[0] << 8 | RANGER_DATA[1]);

       // Transfer 2 bytes of data to the distance in cm.
    Ping_Ranger();
}

void Drive_Motor_Init(void)
{
    // This function will set the drive motor to be neutral for about
       1s.
    DRIVE_MOTOR_PW = DRIVE_MOTOR_NEUT;
    Drive_Motor(); // Turn the drive motor on to neutral.
    Wait_For_1s();
}

void Steering_Servo(void)
{
    // This function will control the steering servo.
    PCA0CPL0 = 0xFFFF - STEERING_SERVO_PW;
    PCA0CPH0 = (0xFFFF - STEERING_SERVO_PW) >> 8; //update servo
       command.
}

void Steering_Servo_Init(void)
{
    // This function will set the drive motor to be neutral for about
       1s.
    STEERING_SERVO_PW = STEERING_SERVO_NEUT;
    Steering_Servo(); // Turn the drive motor on to neutral.
    Wait_For_1s();
}
```

```
void Direction_Controller(void)
{
    // This function will read the compass and set the PW for the
       steering servo.
    Read_Compass();
    HEADING_ERROR = TARGET_HEADING - HEADING; //calculate error
    if (HEADING_ERROR > 1800)                //if error larger than
       simi circle
    {
        HEADING_ERROR -= 3600;
    }                             //turn by opposite direction
    else if (HEADING_ERROR < -1800) //if error larger than simi circle
    {
        HEADING_ERROR += 3600;
    } //turn by opposite direction
    if (DRIVE_MOTOR_PW < DRIVE_MOTOR_NEUT)
    {
        HEADING_ERROR = 0 - HEADING_ERROR;
    }
    STEERING_SERVO_PW = HEADING_ERROR * 5 / 11 + STEERING_SERVO_NEUT;
       // 500 / 1800 = 3/8
    if (STEERING_SERVO_PW > STEERING_SERVO_RIGHT)
    {
        STEERING_SERVO_PW = STEERING_SERVO_RIGHT; //limit SERVO_PW to
           PW_RIGHT
    }
    if (STEERING_SERVO_PW < STEERING_SERVO_LEFT)
    {
        STEERING_SERVO_PW = STEERING_SERVO_LEFT; //limit SERVO_PW to
           PW_LEFT
    }
}

void Read_Compass(void)
{
    // This function will read the data from the compass.
    i2c_read_data(COMPASS_ADDRESS, 2, COMPASS_DATA, 2);
                       // Read 2 byte starting at REG 2.
    HEADING = (((unsigned int)COMPASS_DATA[0] << 8) | COMPASS_DATA[1])
       ; // Convert the data to decimal format.
}

void Read_AD_Input(void)
{
    // This function will read the voltage input and output the
       AD_value.
    AMX1SL = POT_ADDRESS;     // Set P1.4 as the analog input for ADC1
    ADC1CN = ADC1CN & ~0x20; // Clear the "Conversion Completed" flag
    ADC1CN = ADC1CN | 0x10;   // Initiate A/D conversion
    while ((ADC1CN & 0x20) == 0x00)
        ;             // Wait for conversion to complete
    AD_VALUE = ADC1; // Set the AD_VALUE.
}

void POT_Reader(void)
```

```
{
    // This function will use the AD_VALUE and change the max and min
       of DRIVE MOTOR PW.
    Read_AD_Input();

       // Read the data from POT.
    DRIVE_MOTOR_MAX = DRIVE_MOTOR_NEUT + (TARGET_DRIVE_MOTOR_GAIN * 10
        / 255 * AD_VALUE / 10); // Calculate the max of drive motor.
    DRIVE_MOTOR_MIN = DRIVE_MOTOR_NEUT - (TARGET_DRIVE_MOTOR_GAIN * 10
        / 255 * AD_VALUE / 10); // Calculate the min of drive motor.
    STEERING_SERVO_RIGHT = STEERING_SERVO_NEUT + (
       TARGET_STEERING_SERVO_GAIN * 10 / 255 * AD_VALUE / 10);
    // Calculate the right PW for the steering servo.
    STEERING_SERVO_LEFT = STEERING_SERVO_NEUT - (
       TARGET_STEERING_SERVO_GAIN * 10 / 255 * AD_VALUE / 10);
    // Calculate the left PW for the steering servo.
    printf("%d,%d,%d,%d.\r\n", DRIVE_MOTOR_MAX, DRIVE_MOTOR_MIN,
       STEERING_SERVO_RIGHT, STEERING_SERVO_LEFT);
}

void Read_Keypad(void)
{
    lcd_clear();
    lcd_print("Enter␣required␣data.");
    lcd_clear();
    // This function will read the input from the keypad.
    lcd_print("Desired␣direction␣to␣east␣pm␣30␣degree.\n");
    TARGET_HEADING = kpd_input(1); // Get the target heading from the
```

```
        keypad.
    lcd_clear();
    while (TARGET_HEADING < 600 || TARGET_HEADING > 1200)
    {
        // Prompt the user to change the input value if it is not
           satisfied.
        lcd_print("Desired␣direction␣is␣in␣east␣pm␣30␣degree.\n");
        TARGET_HEADING = kpd_input(1);
        lcd_clear();
    }
    printf("TARGET␣HEADING:␣%d.\r\n", TARGET_HEADING);
    lcd_print("Set␣the␣Steering␣servo␣gain.\n");
    TARGET_STEERING_SERVO_GAIN = kpd_input(1); // Get the
       TARGET_STEERING_SERVO_GAIN from the keypad
    printf("Steering␣servo␣gain:␣%d.\r\n", TARGET_STEERING_SERVO_GAIN)
       ;
    lcd_clear();
    lcd_print("Set␣the␣drive␣motor␣gain.\n");
    TARGET_DRIVE_MOTOR_GAIN = kpd_input(1); // Get the
       TARGET_DRIVE_MOTOR_GAIN from the keypad
    printf("Drive␣motor␣gain:␣%d.\r\n", TARGET_DRIVE_MOTOR_GAIN);
    lcd_clear();
    lcd_print("Turn␣SS␣on␣to␣run␣the␣code.");
}

void Flight_Recorder(void)
{
    // This function will output the data to the PuTTY.
    // If the OUTPUT_MODE is 1, it will be output to .csv mode.
    // Else, user-friendly mode.
    if (OUTPUT_MODE != 1)
    {
        // Output to user-friendly UI.
        printf("***BEGIN***\r\n");
        printf("TIME:␣%d.\r\n", TIME);
        printf("DISTANCE:␣%d,␣HEADING:␣%d.\r\n", DISTANCE, HEADING);
        printf("AD_VALUE:␣%d.\r\n", AD_VALUE);
        printf("DRIVE_MOTOR_PW:␣%d,␣STEERING_SERVO_PW:␣%d.\r\n",
           DRIVE_MOTOR_PW, STEERING_SERVO_PW);
        printf("TARGET␣HEADING:␣%d.\r\n", TARGET_HEADING);
        printf("****END****\r\n");
    }
    else
    {
        // Output to .csv format
        printf("%d,%d,%d,%d,%d,%d,%d\r\n", TIME, DISTANCE, HEADING,
           AD_VALUE, DRIVE_MOTOR_PW, STEERING_SERVO_PW, TARGET_HEADING
           );
    }
}

unsigned int Is_SS_On(void)
{
    // This function will return the status of the slide switch.
    if (!SS)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void Turn_BILED_Green(void)
{
    // This function will turn BILED Green
    BILED0 = 0;
    BILED1 = 1;
}

void Turn_BILED_Red(void)
{
    // This function will turn BILED Red
    BILED0 = 1;
    BILED1 = 0;
}

void Turn_BILED_Off(void)
{
    // This function will turn BILED off
    BILED0 = 1;
    BILED1 = 1;
}
```

```
void Port_Init(void)
{
    P1MDIN &= 0xEF;  // Set P1.4 to analog input.
    P1MDOUT &= 0xEF; // Set P1.4 to input mode.
    P1MDOUT |= 0x0D; // Set P1.0, P1.2, P1.3 to output.
    P1 |= ~0xEF;     // Set P1.4 to high impedance.
    P3MDOUT &= 0xBF; // Set P3.6 to input mode.
    P3MDOUT |= 0x03; // Set P3.0, P3.1 to output mode.
    P3 |= ~0xBF;     // Set P3.6 to high impedance.
}

void PCA_Init(void)
{
    PCA0MD = 0x81;    // Enable SYSCLK/12 and enable interrupt.
    PCA0CPM0 = 0xC2;  // Enable CCM0 17bit PWM.
    PCA0CPM2 = 0xC2;  // Enable CCM2 16bit PWM.
    PCA0CN = 0x40;    // Enable PCA counter.
    PCA0 = PCA_START; // For 20ms period.
}

void XBR0_Init(void)
{
    XBR0 = 0x27; // Configure crossbar with UART, SPI, SMBus, and CEX
       channels.
}

void Interrupt_Init(void)
```

```
{
    EA = 1;       // Enable general interrupt.
    EIE1 |= 0x08; // Enable PCA overflow interrupts.
}

void SMB_Init()
{
    SMB0CR = 0x93; // Set the clock frequency to be 100kHz.
    ENSMB = 1;     // Enable SMB.
}

void ADC_Init(void)
{
    REF0CN = 0x03;  // Confi

gure ADC1 to use VREF.
    ADC1CN = 0x80;  // Set a gain of 1.
    ADC1CF |= 0x01; // Enable ADC1.
}

void PCA_ISR(void) __interrupt 9
{
    TIME++;                // Increment the time.
    PCA_COUNTER++;         // Increment the PCA_COUNTER.
    RANGER_READ_COUNTER++; // Increment the RANGER_READ_COUNTER.
    if (!SS)
    {
        Read_Compass();
        //Direction_Controller();
    }
    if (TIME % 10 == 0 && !SS)
    {
        // Output the data to PuTTY every 1s.
        Read_Ranger();
        Flight_Recorder();
    }
    if (CF)
    {
        CF = 0;             // Clear overflow flag.
        PCA0 = PCA_START; // Start count for 20 ms period.
    }
    PCA0CN &= 0x40; // Handle other PCA0 overflows.
}

void Reset_PCA_Counter(void)
{
    // This function reset the PCA_COUNTER.
    PCA_COUNTER = 0;
}

void Wait_For_1s(void)
{
    // WARNING: USING THIS FUNCTION WILL CLEAR PCA_COUNTER.
    // Wait for 1s.
    Reset_PCA_Counter();
    while (PCA_COUNTER < 51)
        ; // Wait for 1s.
    Reset_PCA_Counter();
}

void Troubleshooter(void)
{
    unsigned int Test = 0;
    // This is the test function for the circuit.
    // First test the BILED.
    Turn_BILED_Off();
    Turn_BILED_Green();
    Wait_For_1s();
    Turn_BILED_Red();
    Wait_For_1s();
    Turn_BILED_Off();

    // Then test the drive motor.
    Reset_PCA_Counter();
    DRIVE_MOTOR_PW = DRIVE_MOTOR_MAX;
    while (PCA_COUNTER < 51)
    {
        Drive_Motor();
    }
    Reset_PCA_Counter();
    DRIVE_MOTOR_PW = DRIVE_MOTOR_NEUT;
    while (PCA_COUNTER < 51)
    {
        Drive_Motor();
    }
```

```
Reset_PCA_Counter();
DRIVE_MOTOR_PW = DRIVE_MOTOR_MIN;
while (PCA_COUNTER < 51)
{
    Drive_Motor();
}
Reset_PCA_Counter();
DRIVE_MOTOR_PW = DRIVE_MOTOR_NEUT;
while (PCA_COUNTER < 51)
{
    Drive_Motor();
}
Reset_PCA_Counter();

// Then test the steering servo
STEERING_SERVO_PW = STEERING_SERVO_LEFT;
while (PCA_COUNTER < 51)
{
    Steering_Servo();
}
Reset_PCA_Counter();
STEERING_SERVO_PW = STEERING_SERVO_NEUT;
while (PCA_COUNTER < 51)
{
    Steering_Servo();
}
Reset_PCA_Counter();
STEERING_SERVO_PW = STEERING_SERVO_RIGHT;
```

```
while (PCA_COUNTER < 51)
{
    Steering_Servo();
}
Reset_PCA_Counter();
STEERING_SERVO_PW = STEERING_SERVO_NEUT;
while (PCA_COUNTER < 51)
{
    Steering_Servo();
}
Reset_PCA_Counter();

// Try to test the ranger and compass.
DISTANCE = 0;
HEADING = 0;
Read_Ranger();
printf("DISTANCE:␣%d.\r\n", DISTANCE);
DISTANCE = 0;
Read_Compass();
printf("HEADING:␣%d.\r\n", HEADING);
HEADING = 0;

// Try to read the POT
Read_AD_Input();
printf("AD_VALUE:␣%d.\r\n", AD_VALUE);

// Test the LCD screen
lcd_clear();
```

```
    lcd_print("LCD␣TESTING.\n");
    Wait_For_1s();
    lcd_clear();

    // Test the keypad
    lcd_print("Input␣test␣code.\n");
    Test = kpd_input(1);
    printf("TEST:␣%d.\r\n", Test);
    lcd_clear();
    Reset_PCA_Counter();
}
```

4 ACADEMIC INTEGRITY CERTIFICATION

All the undersigned hereby acknowledge that all parts of this laboratory exercise and report, other than what was supplied by the course through handouts, code templates and web-based media, have been developed, written, drawn, etc. by the team. The guidelines in the Embedded Control Lab Manual regarding plagiarism and academic integrity have been read, understood, and followed. This applies to all pseudo-code, actual C code, data acquired by the software submitted as part of this report, all plots and tables generated from the data, and any descriptions documenting the work required by the lab procedure. It is understood that any misrepresentations of this policy will result in a failing grade for the course.

5 PARTICIPATION

The following individual members of the team were responsible for (give percentages of involvement)

1. Hardware implementation:

Zhongqi Liu	30%
Yilu Zhou	40%
Z Jin	30%

2. Software implementation:

Zhongqi Liu	20%
Yilu Zhou	40%
Z Jin	40%

3. Data Analysis:

Zhongqi Liu	30%
Yilu Zhou	50%
Z Jin	20%

4. Report Development & Editing:

Zhongqi Liu	25%
Yilu Zhou	50%
Z Jin	25%

The following signatures indicate awareness that the above statements are understood and accurate.

-
-
-