MXEN2003 Final Project Report

Lab Time – Tuesday 9AM

Istvan Savanyo (21492387)

Quinn Brands (21490556)

Introduction

This report was written by Istvan Savanyo and Quinn Brands, who both contributed an equal amount of work into the report. Istvan did the circuit diagrams, calculations and calibrations, flow charts, and the description of the logic employed for the autonomous navigation. Quinn did the brief outline, description of the system, highlighting of innovative features, communication protocols, and the reflection on the demonstration.

The code in appendices A and B was also written both by Istvan and Quinn, who both made an equal number of contributions to the final algorithm.

The Brief:

A laboratory has been flooded with radiation, with several victims trapped inside. An autonomous system must be designed to navigate the area and identify victims inside, to limit potential rescuer's exposure to radiation.

It is expected that the system must navigate a simple maze-like area on its own due to a localised radio blackout, in addition to being remotely controlled for the rest of the laboratory area. The system must be able to navigate over and around obstacles, including traversing over small inclines and humps.

The System:

The system that has been designed to complete this task consists of a microcontroller-controlled robot that utilises range sensors, a servo motor and driving motors. It uses an X-Bee to wirelessly communicate to a nearby controller, which is used to control the robot.

The system uses a H-bridge to drive the main motors, which receives input from the joystick on the controller to smoothly drive it around. The three range sensors consist of one long-range sensor (up to ~80cm) on the front, and two short-range sensors (up to ~30cm) on the sides, near the front. Their calibrated readings are output continuously to the controller LCD and the serial monitor so the operator can easily see it.

A camera attached to a servo motor serves as the way for the operator to identify victims in the area. The servo is connected to a second joystick on the controller, allowing the operator to turn the camera to find victims easily.

A battery provides the power for the system, and a battery monitor connected to an LED allows operators to see if the battery requires charging or not. The LED will light up if the total battery power falls below 7V.

The system also includes an autonomous mode, which can be toggled via a button on the controller. This autonomous mode takes in the data from the three range sensors to

navigate a linear maze on its own, without any input from an operator. This allows it to navigate the “radio blackout” portion of the laboratory.

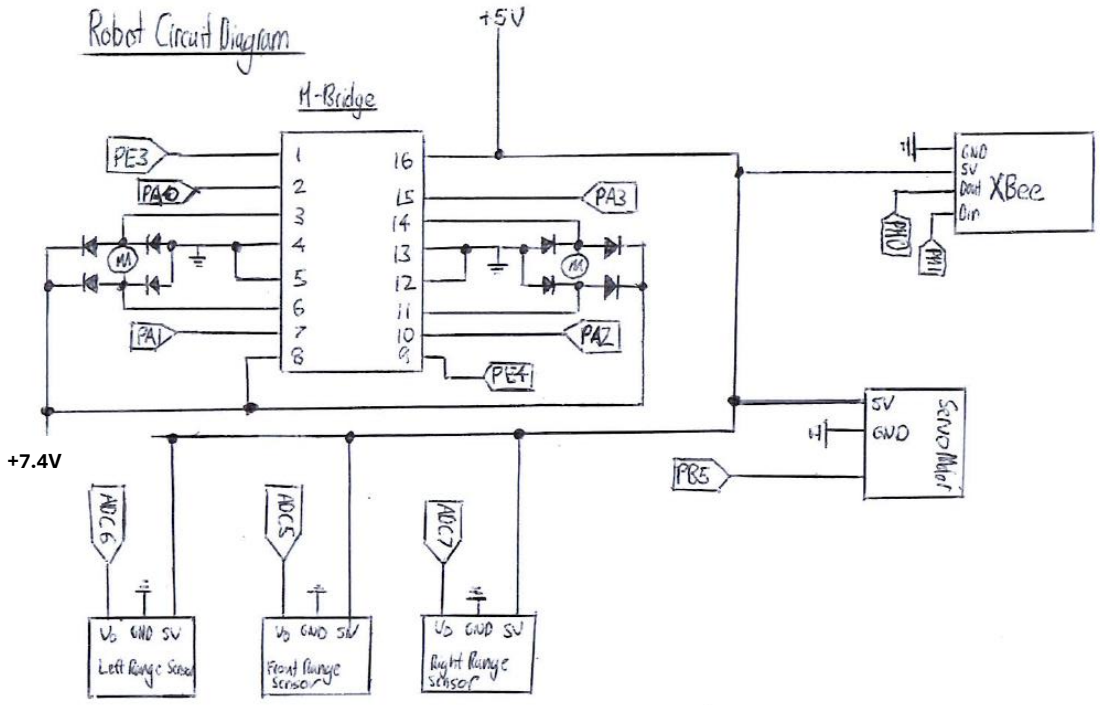
By default, the robot system moves slowly, so that the operator can precisely navigate around obstacles to find and identify victims. However, to help the system get over obstacles, a “Turbo Mode” has also been implemented. The operator can push a separate button on the controller to toggle the mode, providing more power to the primary motors, making the system faster and more powerful, thus allowing it to get over those obstacles.

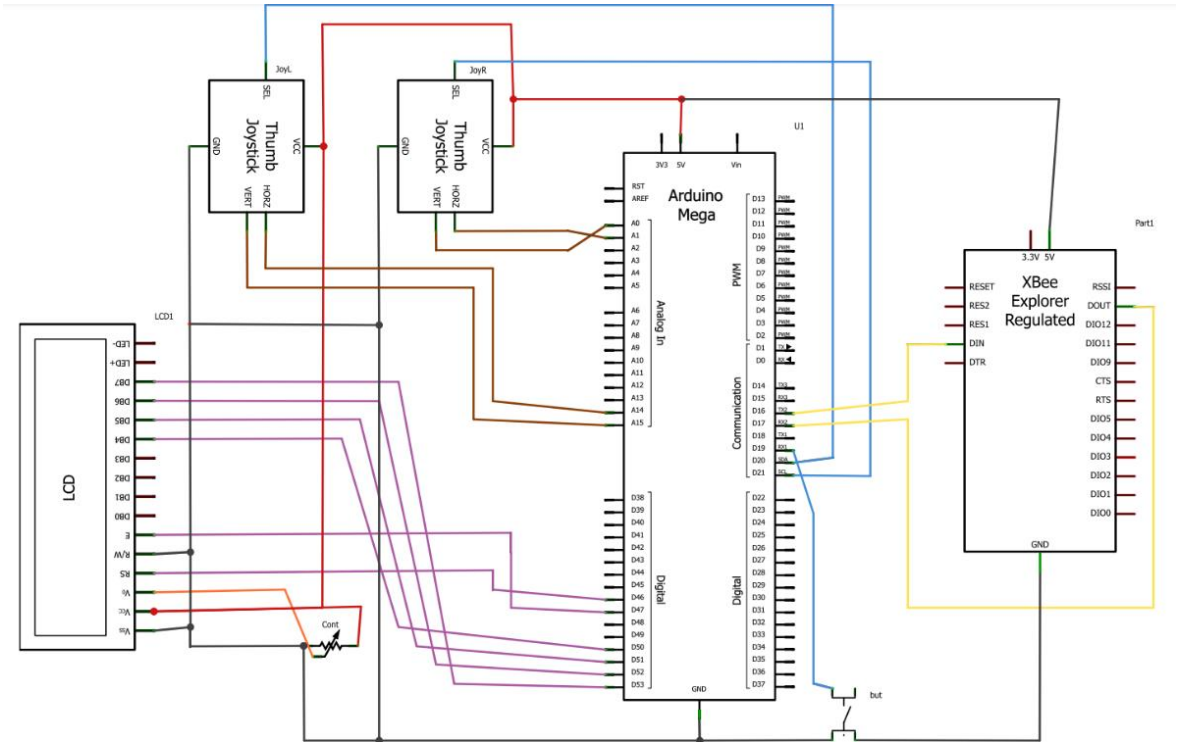
Both the Turbo and Autonomous Modes statuses are displayed on the controller’s LCD, allowing the operator to tell if a mode is on or not easily.

Communication Protocol:

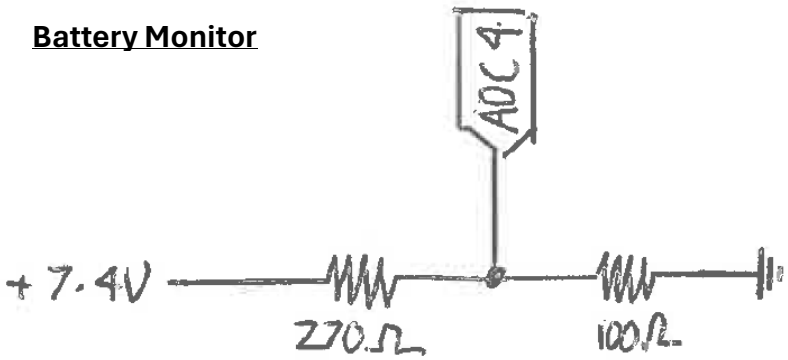
Bit	Robot	Controller
Start	0xFF	0xFF
1	Front Range Sensor (0-253)	Motor Horizontal (0-253)
2	Left Range Sensor (0-253)	Motor Vertical (0-253)
3	Right Range Sensor (0-253)	Servo Control (0-253)
4		Autonomous Mode (0/253)
5		Turbo Mode (0/253)
End	0xFE	0xFE

Circuit Diagrams:





Battery Monitor



Calculations and Calibrations:

Many of the system's features required calculations, or calibrations, to ensure that they worked as intended. One such feature was the battery monitor, which was designed as a voltage divider circuit as the ADC ports on the Arduino only register voltage values in the 0-5V range. The supply voltage from the batteries is 7.4V, so the divider is needed to allow the Arduino to accurately monitor the battery levels.

The battery monitor needs to light up an LED when the supply voltage drops below 7V. Since the Arduino can handle a continuous forward current of 20mA, using the maximum voltage of the batteries, the minimum total resistance needed can be found as:

$$R = \frac{V}{I} = \frac{7.4}{0.02} = 370\Omega \; minimum$$

Using a 270Ω resistor, followed in series by a 100Ω resistor, ensures that all voltages sent to the ADC port will be in the 0-5V range.

$$V_{\max at\ ADC} = 7.4 \times \frac{100}{270 + 100} = 2V \quad (< 5V)$$

The voltage at the ADC when the battery is low (<7V) can be calculated as:

$$V_{\text{low at } ADC} = 7 \times \frac{100}{270 + 100} = 1.89V$$

This voltage can be converted to a 10-bit ADC value (0-1023), using the reference voltage range of 0-5V.

$$ADCval = 1.89 \times \frac{1023}{5} \approx 387$$

This ADC value represents the smallest value that the battery monitor can read at the port before it activates the LED. It represents the point where the battery voltage reaches 7V, and any smaller readings indicates that the battery has dropped below 7V.

There is another resistor that needed to be placed in series with the LED. The value of this resistor was calculated using a supply voltage of 5V, Arduino current of 20mA. We also considered the worst-case scenario, where the LED malfunctions and acts as a short circuit:

$$R = \frac{V}{I} = \frac{5}{0.02} = 250\Omega$$

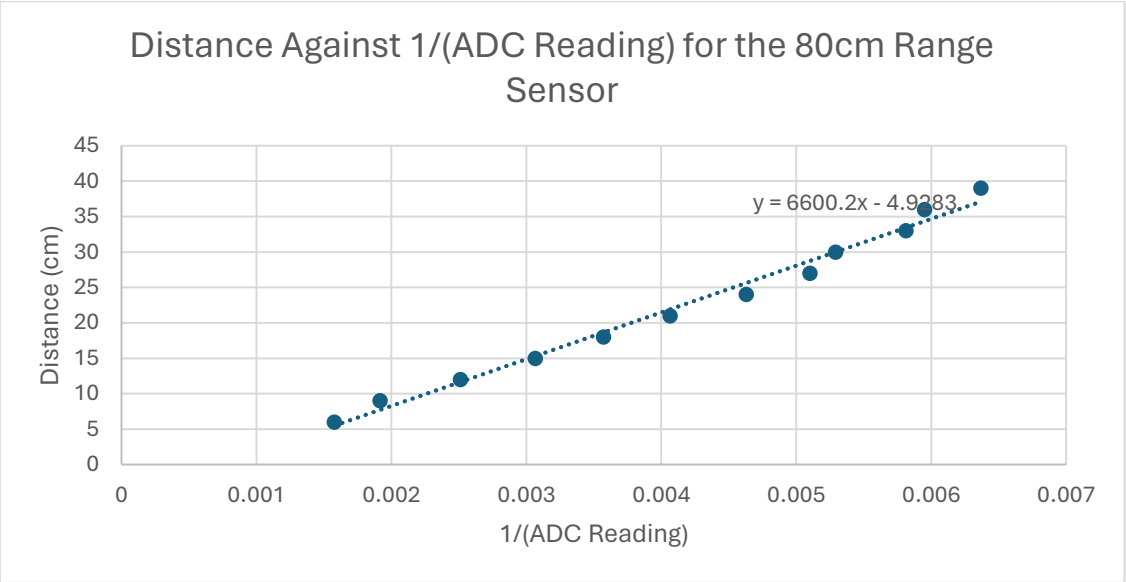
Thus, a 270Ω resistor was used for that LED circuit.

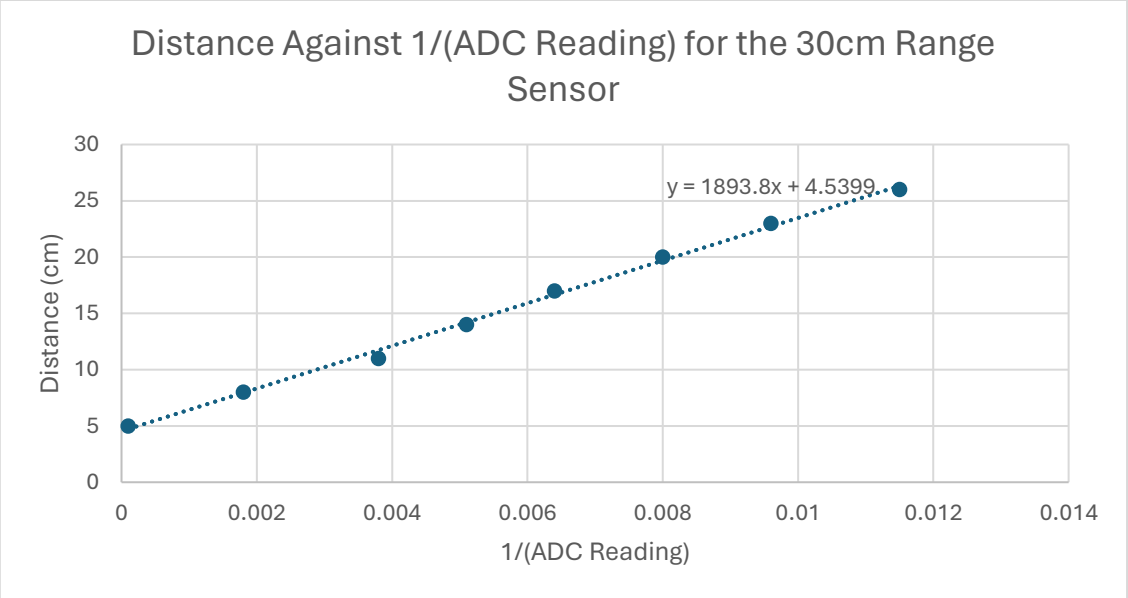
The range sensors needed to be properly calibrated so that the ADC value they returned could be converted into a tangible distance value. This was achieved by recording the

sensors' readings for each distance (taken in 3-5cm intervals) and plotting this on a graph (using the reciprocal of the ADC value on the x-axis). The line of best fit between these data points is the linear equation that can be used to convert ADC to a distance value.

There are two types of range sensors on the robot, two short-range (30cm) sensors on the left and right, and one long-range (80cm) sensor on the front. The calibration process needed to be performed on both types of sensors.

Plotting the gathered data into Excel resulted in the two following graphs:





The long-range sensors' calibration equation was:

$$distance = \frac{6600}{ADCval} - 5$$

Initially the equation gathered for the short-range sensors resulted in distance values being 2cm higher than they should have been. The calibration was slightly after that so that the results were more accurate, resulting in the final calibration equation:

$$distance = \frac{1894}{ADCval} + 2$$

The algorithm contains two PWM clocks, one for the two motors and the other for the servo. Calculations needed to be performed to find the TOP value in the clock. The TOP value is important for defining the base frequency (f_{BASE}) of the PWM signals. Incorrect PWM signals can lead to permanent servo and motor damage, which is unideal.

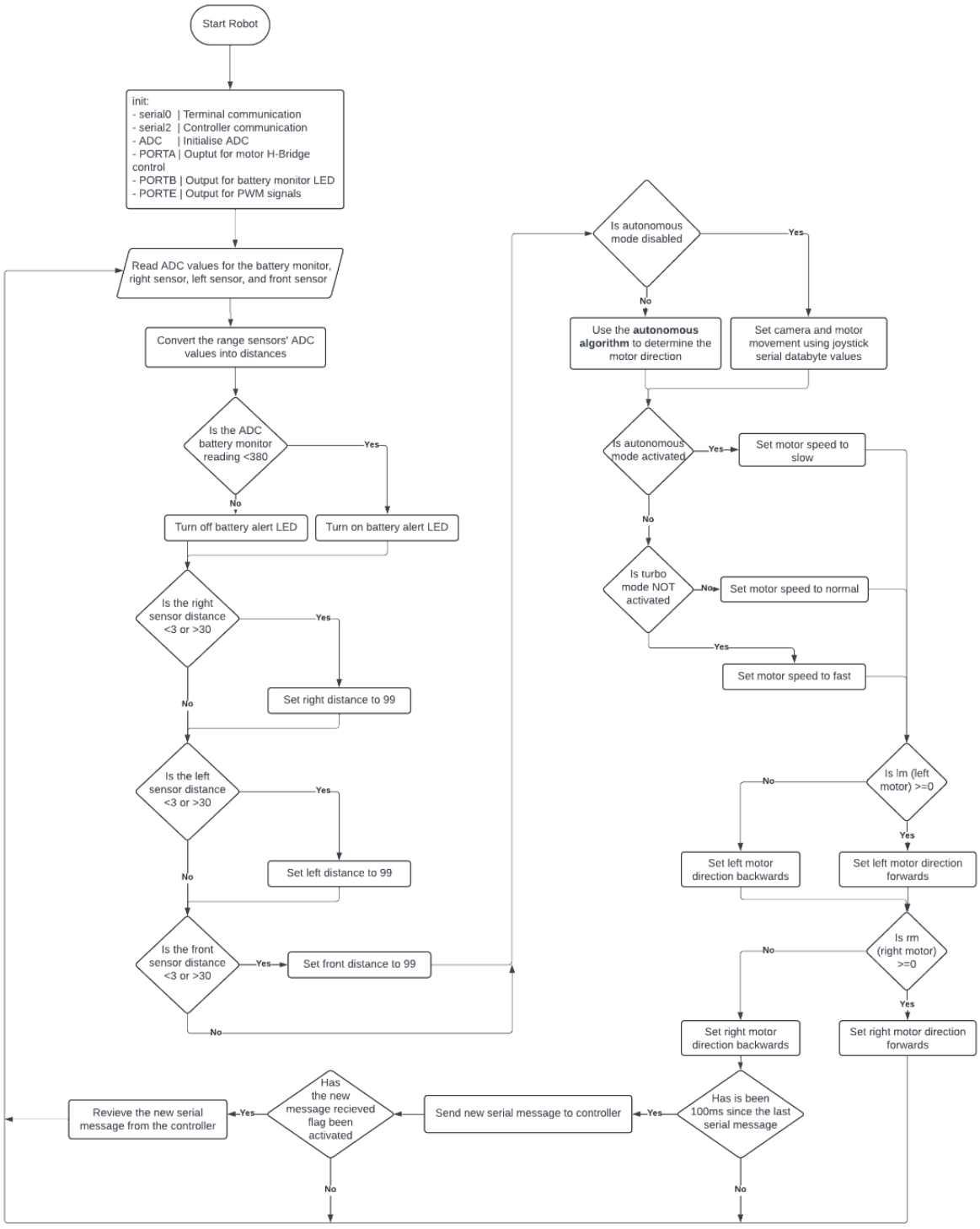
Using the datasheets, and information provided from the MXEN2003 GitHub laboratories, f_{BASE} was determined to be 50Hz for the servo and 250Hz for the motors. Using the equation from Section 17.9 on the ATMega datasheet, and a default PRE value of 8, the TOP values were calculated as follows:

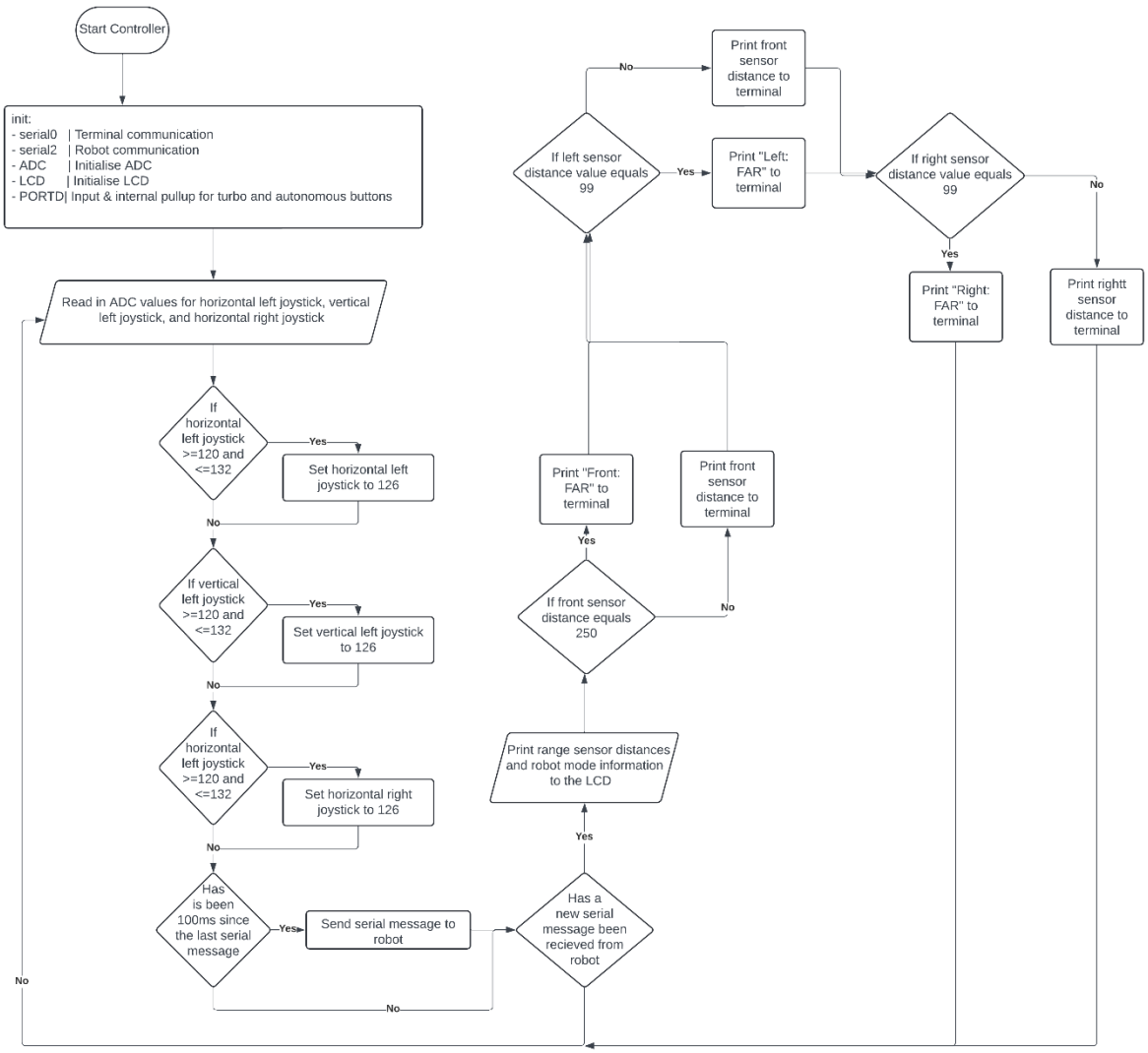
$$TOP_{servo} = \frac{f_{clk}}{2 \times f_{BASE} \times PRE} = \frac{16 \times 10^6}{2 \times 50 \times 8} = 20000$$

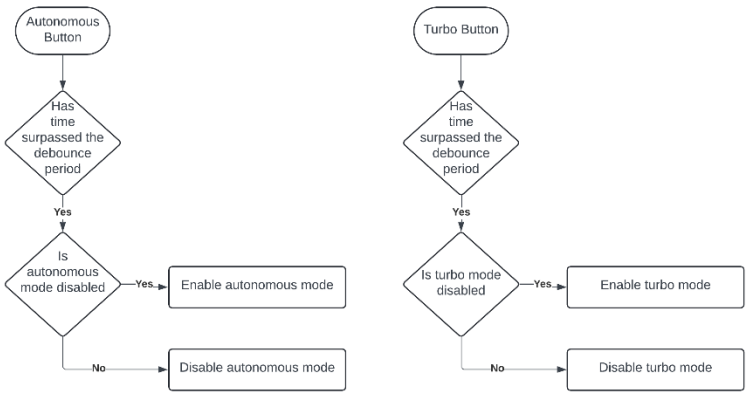
$$TOP_{motors} = \frac{f_{clk}}{2 \times f_{BASE} \times PRE} = \frac{16 \times 10^6}{2 \times 250 \times 8} = 4000$$

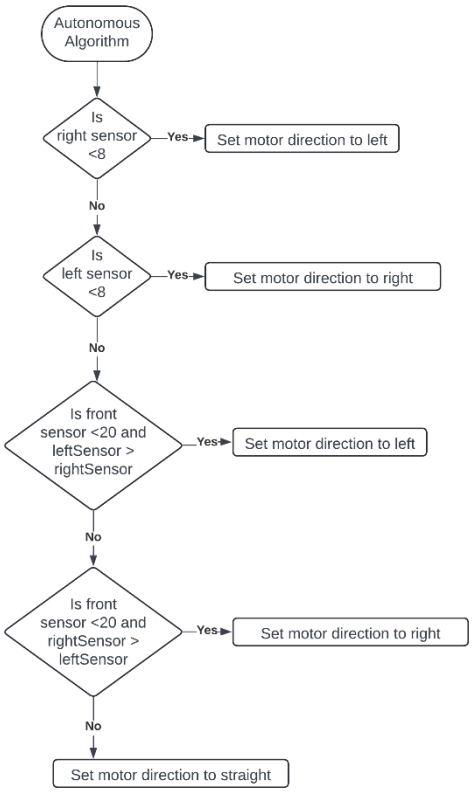
The compare values (OCR1A, OCR3A, OCR3B) were chosen through testing during the labs. The speed of the motors is important for the user operating the robot manually as it defines how smooth the robot feels to control. These compare values were tinkered with until the speed of the robot felt smooth and nice to control.

Flow Charts:


Start Robot
init:
- serial0 | Terminal communication
- serial2 | Controller communication
- ADC | Initialise ADC
- PORTA | Ouptut for motor H-Bridge control
- PORTB | Output for battery monitor LED
- PORTE | Output for PWM signals
Read ADC values for the battery monitor, right sensor, left sensor, and front sensor
Convert the range sensors' ADC values into distances
Is the ADC battery monitor reading <380
Yes
No
Turn off battery alert LED
Turn on battery alert LED
Is the right sensor distance <3 or >30
Yes
Set right distance to 99
No
Is the left sensor distance <3 or >30
Yes
Set left distance to 99
No
Is the front sensor distance <3 or >30
Yes
Set front distance to 99
No
Is autonomous mode disabled
Yes
No
Use the autonomous algorithm to determine the motor direction
Set camera and motor movement using joystick serial databyte values
Is autonomous mode activated
Yes
Set motor speed to slow
No
Is turbo mode NOT activated
No
Set motor speed to normal
Yes
Set motor speed to fast
Is lm (left motor) >=0
No
Yes
Set left motor direction backwards
Set left motor direction forwards
Is rm (right motor) >=0
No
Yes
Set right motor direction backwards
Set right motor direction forwards
Has is been 100ms since the last serial message
Yes
No
Send new serial message to controller
Has the new message recieved flag been activated
Yes
No
Revieve the new serial message from the controller


Start Controller
init:
- serial0 | Terminal communication
- serial2 | Robot communication
- ADC | Initialise ADC
- LCD | Initialise LCD
- PORTD| Input & internal pullup for turbo and autonomous buttons
Read in ADC values for horizontal left joystick, vertical left joystick, and horizontal right joystick
If horizontal left joystick >=120 and <=132
Yes
Set horizontal left joystick to 126
No
If vertical left joystick >=120 and <=132
Yes
Set vertical left joystick to 126
No
If horizontal left joystick >=120 and <=132
Yes
Set horizontal right joystick to 126
No
Has is been 100ms since the last serial message
Yes
Send serial message to robot
No
Has a new serial message been recieved from robot
Yes
Print range sensor distances and robot mode information to the LCD
If front sensor distance equals 250
Yes
Print "Front: FAR" to terminal
No
Print front sensor distance to terminal
If left sensor distance value equals 99
Yes
Print "Left: FAR" to terminal
No
Print front sensor distance to terminal
If right sensor distance value equals 99
Yes
Print "Right: FAR" to terminal
No
Print rightt sensor distance to terminal
No


Autonomous Button
Has time surpassed the debounce period
Yes
Is autonomous mode disabled
Yes
Enable autonomous mode
No
Disable autonomous mode
Turbo Button
Has time surpassed the debounce period
Yes
Is turbo mode disabled
Yes
Enable turbo mode
No
Disable turbo mode



Innovative Feature – Turbo Mode:

We decided to make the robot's motors drive slowly, allowing for precision control through the autonomous section and around obstacles. It gives the operator greater control when needing to find victim tags hidden in hard-to-see places, when small adjustments are preferred to make sure the camera can see the full tag.

However, when testing was completed, it was discovered that these lower powered motors did not have the power to get the robot over some of the various obstacles and ramps it was required to go over. Thus, we decided to implement a "Turbo Mode", which can be toggled on or off to give the motors more power, thereby allowing it to get over these obstacles.

The button to toggle Turbo Mode is the button on the controller linked to the left joystick, which is the one used to control the motors. It sends this via communication bit five to the robot, which then utilises a higher power setting for the motors until Turbo Mode is turned off.

Demonstration Performance Reflection & Improvements:

In the demonstration, the autonomous mode worked flawlessly, successfully getting the robot through the linear maze without any issues. However, we missed a couple of the victim tags in the autonomous section due to the camera not facing the correct way. To

help alleviate this issue, a small and slow oscillation of the camera could've been implemented to help get these tags.

Additionally, an error made in the initial stage of the course was Istvan's body blocking the signal from the X-Bee, resulting in the autonomous mode not being paused at the correct moment, which lost us the first of the victim tags.

The camera (attached to the servo motor) was also a bit too sensitive and moved faster in one direction than the other. It also was hard to centre the camera, resulting in the robot being driven into obstacles because the camera couldn't see them properly.

To improve this, the servo needed to be centred on a specific PWM pulse length (or the current centre found) to equalise the speed between each direction. By utilising the remaining button on the right joystick on the controller, a method to centre the servo could also be implemented. The right joystick is already used to control the servo, so it would make logical sense to use that button for centring the camera. This would make manual driving easier after adjusting the camera to identify a victim tag.

References:

- MXEN2003 GitHub Laboratories
- ATMega Datasheet
- Long-Range Sensor Datasheet
- Short-Range Sensor

Appendix A – Controller.c Code

```
/*************************************************************************
*   Author(s):   Istvan Savanyo (21492387), Quinn Brands (21490556)
*   File:        Controller.c
*   Description: Code for the CONTROLLER Arduino for the Recue Robot Project
*   References:  Algorithm is built upon the original gitHub code supplied
*                in the MXEN2003 labs.
*************************************************************************/

#include "Controller.h"
# define DEBOUNCE_PERIOD 200

static char serial_string[200] = {0};                                   // String for
serial communication.
volatile uint8_t dataByte1=0, dataByte2=0, dataByte3=0, dataByte4=0;        // The recieve
dataByte variables.
volatile bool new_message_received_flag=false;                          // Boolean
flag for recieving a message.
volatile uint8_t autonomousMode=0,turboMode=0;  // Autonomous activator, 0 is OFF, 253 is
ON

int main(void)
{
    /*  Initialisation  */
    serial0_init();     // Initialise terminal communication with PC.
    serial2_init();     // Initialise serial communication with the robot's arduino.
    adc_init();         // Analogue-to-Digital converter initialisation.
    lcd_init();         // LCD Screen initialisation.
    _delay_ms(20);      // 20ms delay.
    uint8_t sendDataByte1=0, sendDataByte2=0, sendDataByte3=0, sendDataByte4=0,
sendDataByte5=0;    // The sent dataByte variables.
    uint8_t frontSensor=0,leftSensor=0,rightSensor=0;
    int16_t cameraJoyHor=0, motorJoyHor=0, motorJoyVer=0;                       //
Joystick input variables.
    uint32_t current_ms=0, last_send_ms=0;                                  //
Variables used for timing the serial send.
    UCSR2B |= (1 << RXCIE2);    // Enable the USART Receive Complete interrupt
(USART_RXC).

    DDRD = 0;                       // Put PORTD into input mode.
    PORTD |= (1<<PD2)|(1<<PD0);     // Enable internal pullup on PD2 (RXD1)

    EICRA |= (1<<ISC01)|(1<<ISC11)|(1<<ISC21);
    EIMSK |= (1<<INT0)|(1<<INT1)|(1<<INT2);

    milliseconds_init();        // Microsecond timer initialisation (Timer 5 on ATMega).
```

```
    sei();                      // Enable interrupts.



    /*  Start of Looping Algorithm  */
    while(1)
    {
        /*  Sending Section */
        current_ms = milliseconds_now();        // Gets current time on the micros timer.
        if(current_ms-last_send_ms >= 100)      // Sending rate controlled here one
message every 100ms (10Hz)
        {
            /*  Reading/Modifying Joystick Input  */

            motorJoyHor = adc_read(14)*0.2475;          // Modified to be within a range
of 253.
            motorJoyVer = 253-adc_read(15)*0.2475;      // Modified so pushing UP is MAX
(253). Joystick is flipped on controller.
            cameraJoyHor = adc_read(1)*0.2475;          // Modified to be within a range
of 253.

            if(motorJoyHor >= 120 && motorJoyHor <= 132){motorJoyHor=126;}  // If between
124 and 128, assume joystick is in middle position.
            if(motorJoyVer >= 120 && motorJoyVer <= 132){motorJoyVer=126;}
            if(cameraJoyHor >= 120 && cameraJoyHor <= 132){cameraJoyHor=126;}
            /* Debugging Serial Print */

            /*  Setting SEND Databytes */
            sendDataByte1 = motorJoyHor;        // Setting send databyes
            sendDataByte2 = motorJoyVer;
            sendDataByte3 = cameraJoyHor;
            sendDataByte4 = autonomousMode;
            sendDataByte5 = turboMode;

            /* Sending the Databyes */
            last_send_ms = current_ms;
            serial2_write_byte(0xFF);           // Send start byte = 255
            serial2_write_byte(sendDataByte1);  // Send first data byte:  Horizontal Motor
Joystick
            serial2_write_byte(sendDataByte2);  // Send second parameter: Vertical Motor
Joystick
            serial2_write_byte(sendDataByte3);  // Send third data byte:  Horizontal
Camera Joystick
            serial2_write_byte(sendDataByte4);  // Send fourth parameter: Autonomous
BUTTON Trigger
            serial2_write_byte(sendDataByte5);  // Send fifth parameter:  TURBO BUTTON
Trigger
            serial2_write_byte(0xFE);           // Send stop byte = 254
        }
```

```
        //if a new byte has been received
        if(new_message_received_flag)
        {
            frontSensor = dataByte1;
            leftSensor = dataByte2;
            rightSensor = dataByte3;


            lcd_goto(0);
            sprintf(serial_string,"F=%3u L=%2u R=%2u",frontSensor,leftSensor,rightSensor);
            lcd_puts(serial_string);
            lcd_goto(0x40);
            sprintf(serial_string,"T:%u  A:%u",turboMode,autonomousMode);
            lcd_puts(serial_string);


            if(frontSensor==250){
                serial0_print_string("Front: FAR     ");
            }
            else{
                sprintf(serial_string,"Front: %3ucm   ",frontSensor);
                serial0_print_string(serial_string);
            }
            if(leftSensor==99){
                serial0_print_string("Left: FAR     ");
            }
            else{
                sprintf(serial_string,"Left: %3ucm   ",leftSensor);
                serial0_print_string(serial_string);
            }
            if(rightSensor==99){
                serial0_print_string("Right: FAR     \n");
            }
            else{
                sprintf(serial_string,"Right: %3ucm   \n",rightSensor);
                serial0_print_string(serial_string);
            }

            if(dataByte1 != 0){
            }
            else{
            }
            new_message_received_flag=false;    // set the flag back to false
        }
    }
    return(1);
} //end main
```

```
static uint32_t previousTime = 0;
ISR(INT2_vect){      // Toggles AUTONOMOUS MODE
    uint32_t currentTime = milliseconds_now();

    if( (currentTime - previousTime) > DEBOUNCE_PERIOD ){   // 100ms button debounce
        if( autonomousMode == 0 ){      // If NOT autonomous, turn it on.
            autonomousMode = 1;
        }
        else{                           // Else, turn it off
            autonomousMode = 0;
        }
        previousTime = currentTime;
    }
}
ISR(INT1_vect){      // Toggles TURBO MODE
    uint32_t currentTime = milliseconds_now();

    if( (currentTime - previousTime) > DEBOUNCE_PERIOD ){   // 100ms button debounce
        if( turboMode == 0 ){       // If NOT TURBO, turn it on.
            turboMode = 1;
        }
        else{                           // Else, turn it off
            turboMode = 0;
        }
        previousTime = currentTime;
    }
}
ISR(USART2_RX_vect)  // ISR executed whenever a new byte is available in the serial buffer
{
    static uint8_t recvByte1=0, recvByte2=0, recvByte3=0, recvByte4=0;      // data bytes
received
    static uint8_t serial_fsm_state=0;                                  // used in the
serial receive ISR
    uint8_t serial_byte_in = UDR2; //move serial byte into variable

    switch(serial_fsm_state) //switch by the current state
    {
        case 0:
        //do nothing, if check after switch case will find start byte and set
serial_fsm_state to 1
        break;
        case 1: //waiting for first parameter
        recvByte1 = serial_byte_in;
        serial_fsm_state++;
        break;
        case 2: //waiting for second parameter
        recvByte2 = serial_byte_in;
```

```
        serial_fsm_state++;
        break;
        case 3: //waiting for third parameter
        recvByte3 = serial_byte_in;
        serial_fsm_state++;
        break;
        case 4: //waiting for fourth parameter
        recvByte4 = serial_byte_in;
        serial_fsm_state++;
        break;
        case 5: //waiting for stop byte
        if(serial_byte_in == 0xFE) //stop byte
        {
            // now that the stop byte has been received, set a flag so that the
            // main loop can execute the results of the message
            dataByte1 = recvByte1;
            dataByte2 = recvByte2;
            dataByte3 = recvByte3;
            dataByte4 = recvByte4;

            new_message_received_flag=true;
        }
        // if the stop byte is not received, there is an error, so no commands are
implemented
        serial_fsm_state = 0; //do nothing next time except check for start byte (below)
        break;
    }
    if(serial_byte_in == 0xFF) //if start byte is received, we go back to expecting the
first data byte
    {
        serial_fsm_state=1;
    }
}
```

Appendix B – Robot.c Code

```
/*****************************************************************************
*   Author(s):    Istvan Savanyo (21492387), Quinn Brands (21490556)
*   File:         Robot.c
*   Description: Code for the ROBOT Arduino for the Recue Robot Project
*   References:  Algorithm is built upon the original gitHub code supplied
*                in the MXEN2003 labs.
*****************************************************************************/

#include "Robot.h"

static char serial_string[200] = {0};                                   // String for
serial communication.
volatile uint8_t dataByte1=0, dataByte2=0, dataByte3=0, dataByte4=0, dataByte5=0;       //
The recieve dataByte variables.
volatile bool new_message_received_flag=false;                          // Boolean
flag for recieving a message.

static int16_t lm = 0, rm = 0;              // Variables used for MOTOR and SERVO control
static int16_t fc = 0, rc = 0, cam = 126;


int main(void)
{
    /*  Initialisation  */
    serial0_init();     // Initialise terminal communication with PC.
    serial2_init();     // Initialise serial communication with the controller's arduino.
    adc_init();         // Analogue-to-Digital converter initialisation.
    _delay_ms(20);      // 20ms delay.
    uint8_t sendDataByte1=0, sendDataByte2=0, sendDataByte3=0, sendDataByte4=0;     // The
sent dataByte variables.
    uint16_t rightSensorVal=0, leftSensorVal=0, frontSensorVal=0;               // All
3 sensor values.
    uint16_t rightSensorAvg=15, leftSensorAvg=15, frontSensorAvg=15;            //
Average values sensors (default to 15 at start).
    uint8_t autonomousMode=0,turboMode=0;               // Variables for turning on, and
pausing, autonomousMode mode
    uint32_t current_ms=0, last_send_ms=0;                                      //
Variables used for timing the serial send.
    uint16_t batteryADCReadingAvg=485;
    int16_t comp=0,comp2=0;
    UCSR2B |= (1 << RXCIE2); // Enable the USART Receive Complete interrupt (USART_RXC).
    milliseconds_init();     // Microsecond timer initialisation (Timer 5 on ATMega).

    /*  Initialise the PWM for MOTOR Control [Timer 3] */
    TCCR3B |= (1<<WGM33)|(1<<CS31);     // Mode 8, PRE=8.
    TCCR3A |= (1<<COM3A1)|(1<<COM3B1);  // Compare mode (2 compares).
```

```
    TCNT3 = 0;
    ICR3 = 4000;                        // Setting TOP value.
    DDRE |= (1<<PE3)|(1<<PE4);          // Initialising PWM pins as OUTPUT.
    OCR3A = 2000;                       // Setting first compare value.
    OCR3B = 2000;                       // Setting second compare value.

    /*  Initialise the PWM for SERVO Control [Timer 1]  */
    TCCR1B |= (1<<WGM13)|(1<<CS11);     // Mode 8, PRE=8.
    TCCR1A |= (1<<COM1A1);              // Compare mode (2 compares).
    TCNT1 = 0;
    ICR1 = 20000;                       // Setting TOP value.
    DDRB |= (1<<PB5);                   // Initialising PWM pin as OUTPUT.
    OCR1A = 1500;                       // Setting compare value.
    /*  PORT/PIN Initialisation  */
    DDRA = 0xFF;        // For motor H-Bridge control.
    DDRB = 0xFF;        // For battery alert LED
    sei();              // Enable interrupts.


    /*  Start the Looping Algorithm  */
    while(1)
    {

        batteryADCReadingAvg = 0.9*batteryADCReadingAvg + 0.1*adc_read(4);
        if(batteryADCReadingAvg < 380){PORTB |= (1<<PB3);}      // Turn on LED if ADC is
below 395 (below 7V volts).
        else{PORTB &= ~(1<<PB3);}                               // Else turn it off.




        /*  Sending Section  */
        current_ms = milliseconds_now();        // Gets current time on the micros timer.

        rightSensorAvg = rightSensorAvg*9/10 + adc_read(7)/10;  // Get new sensor values
based on average.
        leftSensorAvg = leftSensorAvg*9/10 + adc_read(6)/10;    // Creates smoother/nicer
sensor values.
        frontSensorAvg = frontSensorAvg*9/10 + adc_read(5)/10;

        rightSensorVal = 1894/rightSensorAvg+2;     // Convert from ADC value to distance.
        leftSensorVal = 1894/leftSensorAvg+2;
        frontSensorVal = 6600/frontSensorAvg-5;

        if (rightSensorVal > 30 || rightSensorVal < 3) {rightSensorVal = 99;}   // Sensor
distance of 253 means OUT OF RANGE.
        if (leftSensorVal > 30 || leftSensorVal < 3) {leftSensorVal = 99;}
        if (frontSensorVal > 80 || frontSensorVal < 3) {leftSensorVal = 99;}
```

```
        /*  MOTOR SHIT PISS  */
        if(autonomousMode == 0){
            lm = (fc-126) + (rc-126);
            rm = (fc-126) - (rc-126);
            if(cam >= 124 && cam <= 128){
                OCR1A = 0;
            }
            else{
                OCR1A = 1750-(int32_t)cam*3.953/2;
            }
        }
        else{
            comp=leftSensorVal-rightSensorVal;
            comp2=rightSensorVal-leftSensorVal;
            if (rightSensorVal<8){  //left turn
                sprintf(serial_string,"TOO CLOSE TO LEFT");
                serial0_print_string(serial_string);
                lm = -200;
                rm = -200;
            }
            else if (leftSensorVal<8){  // right turn
                sprintf(serial_string,"TOO CLOSE TO RIGHT");
                serial0_print_string(serial_string);
                lm = 200;
                rm = 200;
            }

            else if( (frontSensorVal<20 && (comp>5)) ){         // left turn
                sprintf(serial_string,"LEFT");
                serial0_print_string(serial_string);
                lm = -200;
                rm = -200;
            }
            else if( (frontSensorVal<20 && (comp2>5))){     // right turn
                sprintf(serial_string,"RIGHT");
                serial0_print_string(serial_string);
                lm = 200;
                rm = 200;
            }
            else{           // Default - Forward
            sprintf(serial_string,"Fpoward");
            serial0_print_string(serial_string);
                lm = 127;
                rm = -127;
            }
        }
        if(autonomousMode!=0){
            OCR3A = (int32_t)abs(lm)*2000/126; //lm speed from magnitude of lm
            OCR3B = (int32_t)abs(rm)*2000/126; //lm speed from magnitude of rm
```

```
        }
        else if(turboMode==0){
            OCR3A = (int32_t)abs(lm)*3000/126; //lm speed from magnitude of lm
            OCR3B = (int32_t)abs(rm)*3000/126; //lm speed from magnitude of rm
        }
        else{
            OCR3A = (int32_t)abs(lm)*20000/126; //lm speed from magnitude of lm
            OCR3B = (int32_t)abs(rm)*20000/126; //lm speed from magnitude of rm
        }
        if(lm>=0){
            PORTA |= (1<<PA0);   // Set direction forwards (if lm positive).
            PORTA &= ~(1<<PA1);
        }
        else{
            PORTA &= ~(1<<PA0);  // Set direction reserse.
            PORTA |= (1<<PA1);
        }

        if(rm>=0){
            PORTA |= (1<<PA2);   // Set direction forwards (if lm positive).
            PORTA &= ~(1<<PA3);
        }
        else{
            PORTA &= ~(1<<PA2);  // Set direction reverse.
            PORTA |= (1<<PA3);
        }



        if(current_ms-last_send_ms >= 100)      // Sending rate controlled here one
message every 100ms (10Hz).
        {
            sendDataByte1 = frontSensorVal;
            sendDataByte2 = leftSensorVal;
            sendDataByte3 = rightSensorVal;
            last_send_ms = current_ms;
            serial2_write_byte(0xFF);           // Send start byte = 255
            serial2_write_byte(sendDataByte1);  // Send first data byte:  Front Sensor
Reading
            serial2_write_byte(sendDataByte2);  // Send second parameter: must be scaled
to the range 0-253
            serial2_write_byte(sendDataByte3);  // Send first data byte: must be scaled to
the range 0-253
            serial2_write_byte(sendDataByte4);  // Send second parameter: must be scaled
to the range 0-253
            serial2_write_byte(0xFE);           // Send stop byte = 254

            sprintf(serial_string,"R %2u L %2u
F%2u\n",rightSensorVal,leftSensorVal,frontSensorVal);
```

```
            serial0_print_string(serial_string);
        }

        //if a new byte has been received
        if(new_message_received_flag)
        {
            fc = dataByte1;
            rc = dataByte2;
            cam = dataByte3;
            autonomousMode = dataByte4;
            turboMode = dataByte5;
            new_message_received_flag=false;    // set the flag back to false
        }
    }
    return(1);
} //end main


ISR(USART2_RX_vect)  // ISR executed whenever a new byte is available in the serial buffer
{
    static uint8_t recvByte1=0, recvByte2=0, recvByte3=0, recvByte4=0, recvByte5=0;     //
data bytes received
    static uint8_t serial_fsm_state=0;                                  // used in the
serial receive ISR
    uint8_t serial_byte_in = UDR2; //move serial byte into variable

    switch(serial_fsm_state) //switch by the current state
    {
        case 0:
        //do nothing, if check after switch case will find start byte and set
serial_fsm_state to 1
        break;
        case 1: //waiting for first parameter
        recvByte1 = serial_byte_in;
        serial_fsm_state++;
        break;
        case 2: //waiting for second parameter
        recvByte2 = serial_byte_in;
        serial_fsm_state++;
        break;
        case 3: //waiting for third parameter
        recvByte3 = serial_byte_in;
        serial_fsm_state++;
        break;
        case 4: //waiting for fourth parameter
        recvByte4 = serial_byte_in;
        serial_fsm_state++;
        break;
        case 5: //waiting for fifth parameter
        recvByte5 = serial_byte_in;
```

```
        serial_fsm_state++;
        break;
        case 6: //waiting for stop byte
        if(serial_byte_in == 0xFE) //stop byte
        {
            // now that the stop byte has been received, set a flag so that the
            // main loop can execute the results of the message
            dataByte1 = recvByte1;
            dataByte2 = recvByte2;
            dataByte3 = recvByte3;
            dataByte4 = recvByte4;
            dataByte5 = recvByte5;

            new_message_received_flag=true;
        }
        // if the stop byte is not received, there is an error, so no commands are
implemented
        serial_fsm_state = 0; //do nothing next time except check for start byte (below)
        break;
    }
    if(serial_byte_in == 0xFF) //if start byte is received, we go back to expecting the
first data byte
    {
        serial_fsm_state=1;
    }
}
```