PROJECT REPORT

Daniel Waits (19779797) & Jason Kim (19794761)

ABSTRACT

This report presents our group's findings, calculations and presentation of all relevant circuit diagram and programming logic utilised to achieve the project task presented towards us.

Mechatronic Microcontroller Project

MXEN2002

Table of Contents

Table of Contents

Table of Figures

1.0 Task description

- As per the project description provided by the course co-ordinator of MXEN2002 we were tasked with the design and build of a semi-autonomous robot. With the equipment provided, i.e.: an Arduino mega board, a robot base plot, two DC motors, two caster wheels, a position servo motor, 2 analogue, 2 sharp short range sensors, 1 sharp long range sensor, 1 h-bridge and a $9\ V$ battery, we can to demonstrate a variety of technical tasks. These technical tasks were based off of an original project prompt that simulated a task for a semi-autonomous robot to navigate a particular area. These tasks were to built a robot that uses a h-bridge to control two DC motors, the circuitry should be neat and compact, code should use serial loopback communication, the robot should achieve smooth differential drive using one joystick, servo motor control using another joystick, calibrated range sensors that are printed to serial and a demonstration of being able to move autonomously in a particular environment.

2.0 Flow chart

2.1 Description of flow charts.

- For the purpose of readability and simplicity this overall flow chart has been broken down across three flow charts. One of the flow charts that outline the overall program functionality, one flow chart that indicates the sequence of events when manual mode is triggered and another flow chart that indicates that sequence of events for when autonomous mode is triggered. These flow charts mimic the sequence of events of the code as seen in appendix item 8.2. These flow charts can be found in the subsequent pages that follow.

2.2 Overall program flow chart

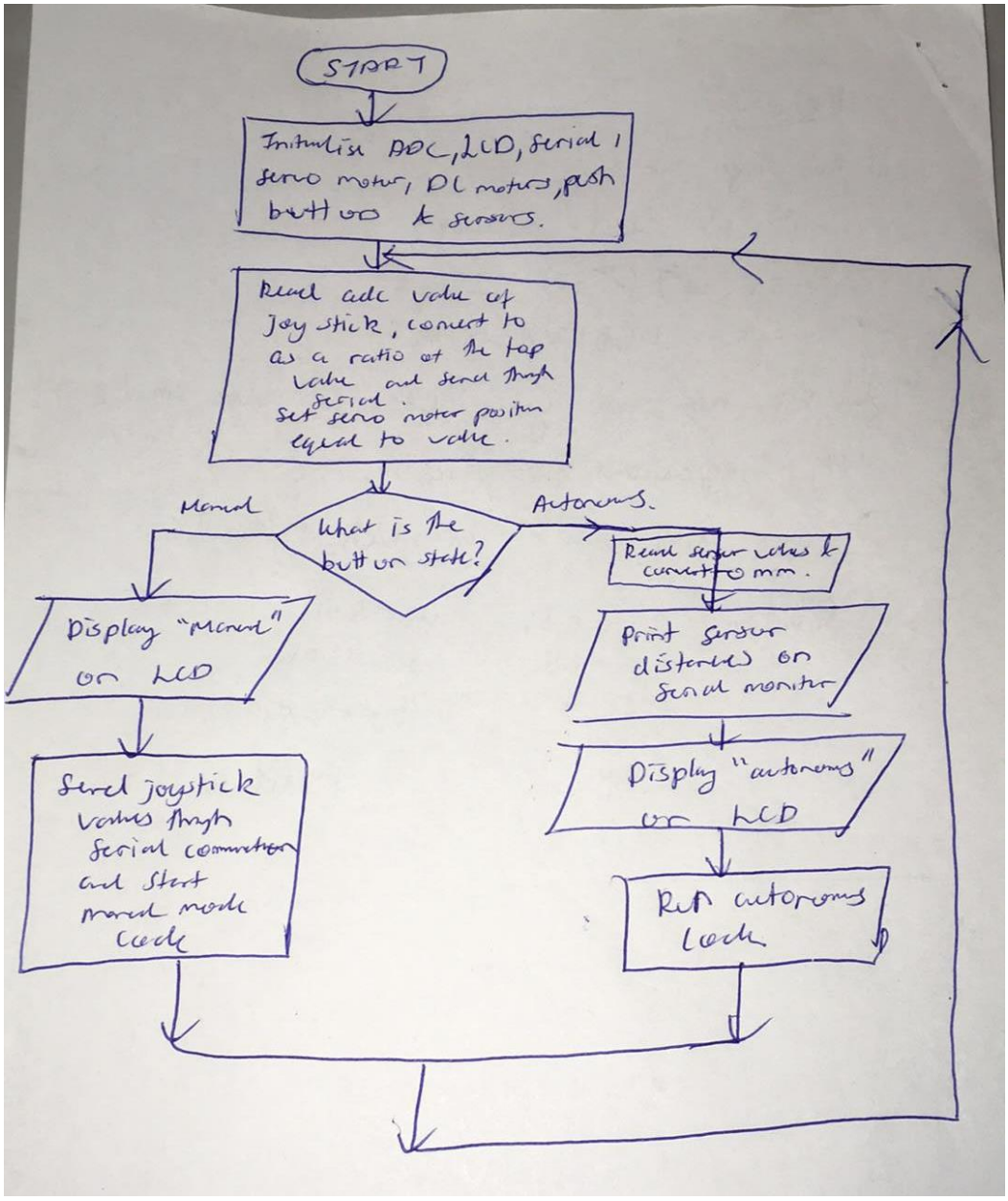


Figure 1: Overall Program Flowchart

2.3 Autonomous mode flow chart

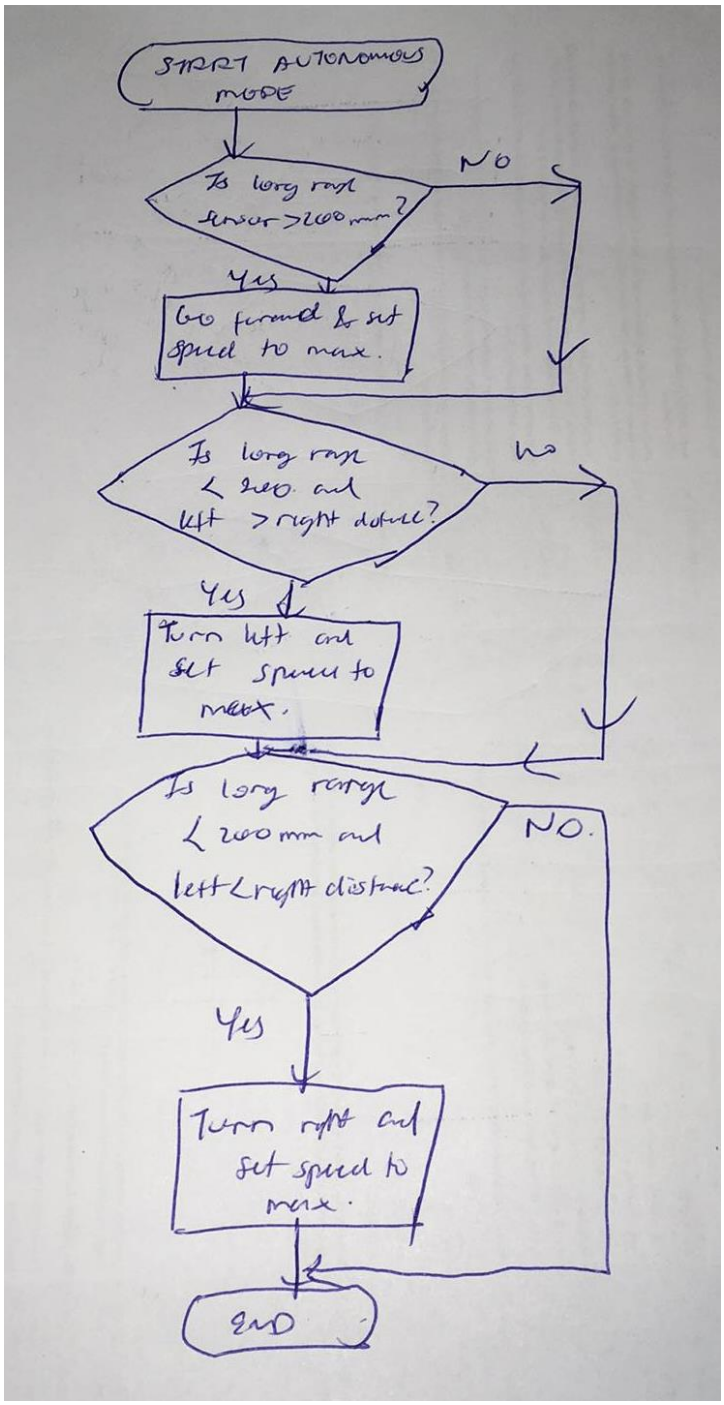


Figure 2: Autonomous Mode Flowchart

2.4 Manual mode flow chart

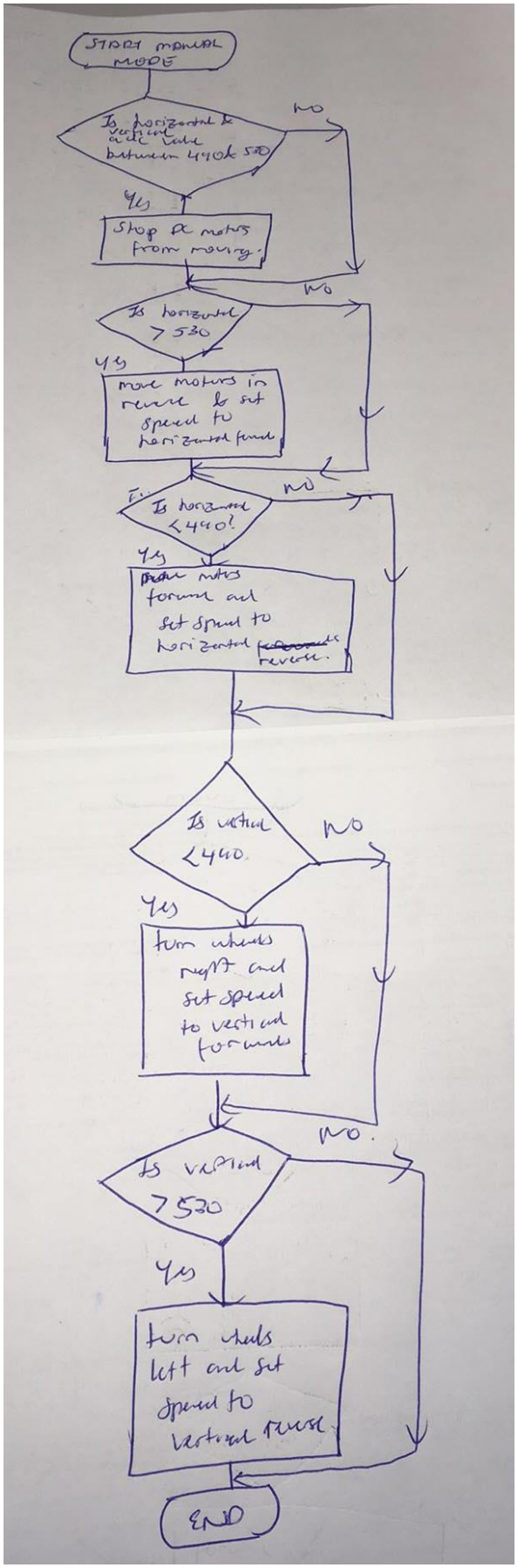


Figure 3: Manual Mode Flowchart

3.0 Autonomous mode

- The purpose of autonomous mode is to allow the robot, to navigate simple terrain by itself, without any user input. This is done through a series of separate cases, that if met, tell the robot what to do. These cases are, going forwards, turning left, and turning right. In order to enter these cases, the robot reads the values from the 3 range sensors. In order to get the robot to go forward, it must have at least 200mm of room in the front and have no objects in the near vicinity on either side. This will continue to happen, until the robot needs to turn. Due to the terrain being very simple and basic, we were given information that we would only have to make turns where there is a wall in front and on either the left or right. This simplifies it so that we can say, if the front range is less than 200mm and the left is considerably less than the right, the robot should turn right. Using these 3 simple conditions, we were able to give our robot, upon pressing the button, simple autonomy.

4.0Manual mode

- Manual mode is the state in which a controller is able to control the differential drive system of the DC motors that are connected the h-bridge. Depending on the joystick positions that robot is able to move accordingly in that direction and the speed is set to as a fraction of the maximum depending on the read joystick value. To achieve this first the analogue value of the horizontal and vertical position of the joy stick are read and these values are used to determine the direction of the motor. To determine the speed of the robot these analogue values are taken down as a percentage of the maximum and minimum analogue values via the equations shown below.
    - $speed_{Horizontal\ reverse} = \left|(adc_{horizontal} - 512) \times \left(\frac{199}{512}\right)\right|$
    - $speed_{horizontal\ forward} = (adc_{horizontal} - 512) \times \left(\frac{199}{512}\right)$
    - $speed_{vertical\ reverse} = \left|(adc_{vertical} - 512) \times \left(\frac{199}{512}\right)\right|$
    - $speed_{vertical\ forward} = (adc_{vertical} - 512) \times \left(\frac{199}{512}\right)$
- As seen in the flow chart for manual mode control these speeds are set as the respective motor speed depending on what quadrant the position of the joystick is currently in. This differential drive is achieved using 6 if statements that represent the 6 different possible states that the joystick position could be in.

5.0 Circuit diagram

5.1 Description of Circuit Diagrams

- The following section of the report outlines all the different types of circuit component that our group utilised in order to achieve the project task. Under each circuit component there contains relevant description of its pin connection, purpose/ role and any relevant calculation pertaining to the component.

5.2 H-bridge / DC Motors

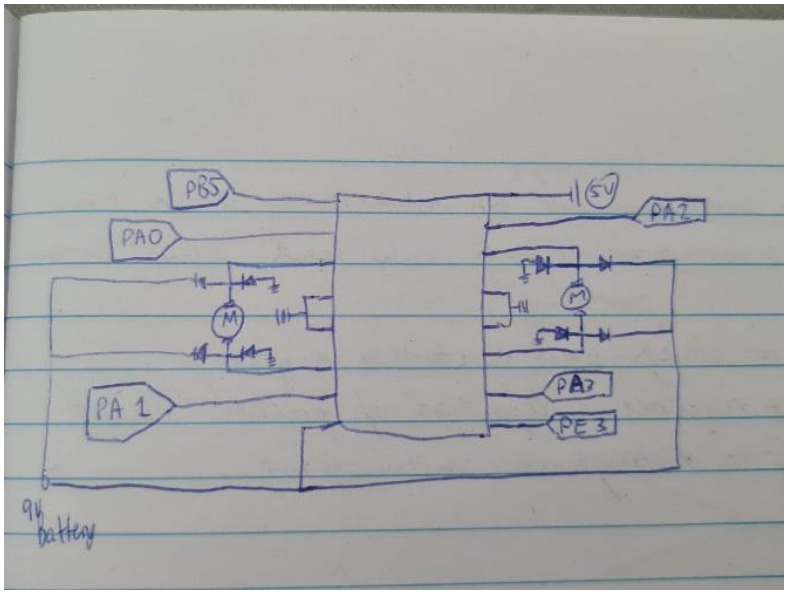


Figure 4: H-Bridge / DC Motors Circuit Diagram

- H-bridge is used to connect both of the motor together, so that they can be controlled individually.
- Motor calculation, using a pre-scalar of 8 and frequency $10\ 000\ Hz$.
  - $Top = \frac{16\times10^6\ Hz}{10\ 00\ Hz\times8} - 1$
  - $\therefore Top = 199$

5.3 Joystick

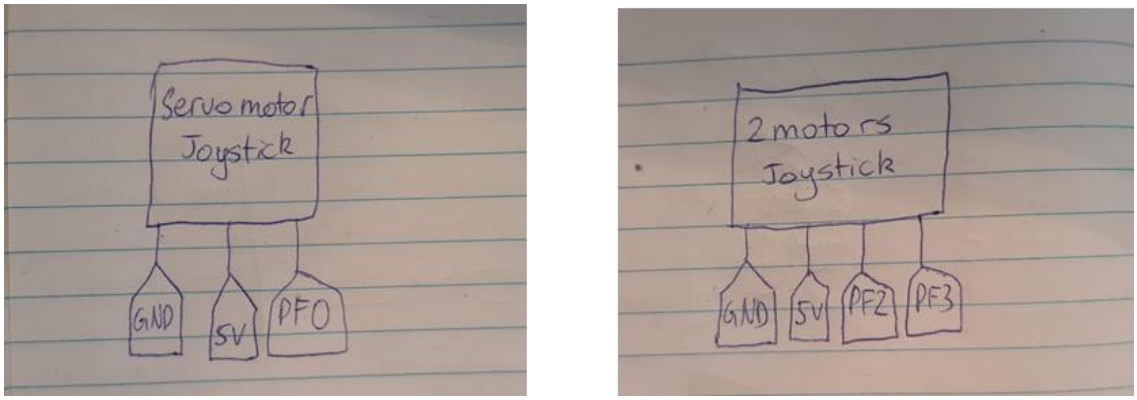


Figure 5: Joystick Circuit Diagrams

- There is one joystick for the 2 motors, vertical and horizontal, and the other one for the servo motor, just horizontal.
- Calculation using $h = horizontal\ adc\ reading$, same calculations for vertical:
  - $Scaled\ forward\ value = (h - 512) \times \frac{199}{512}$
  - $Scaled\ reverses\ value = abs\left((h - 512) \times \left(\frac{199}{512}\right)\right)$
- Calculation for communication protocol, scale 0-253
  - $Scaled\ forward\ value = (h - 512) \times \left(\frac{253}{512}\right)$
  - $Scaled\ reverse\ value\ = abs((h - 512) \times \left(\frac{253}{512}\right)$

5.4 LCD

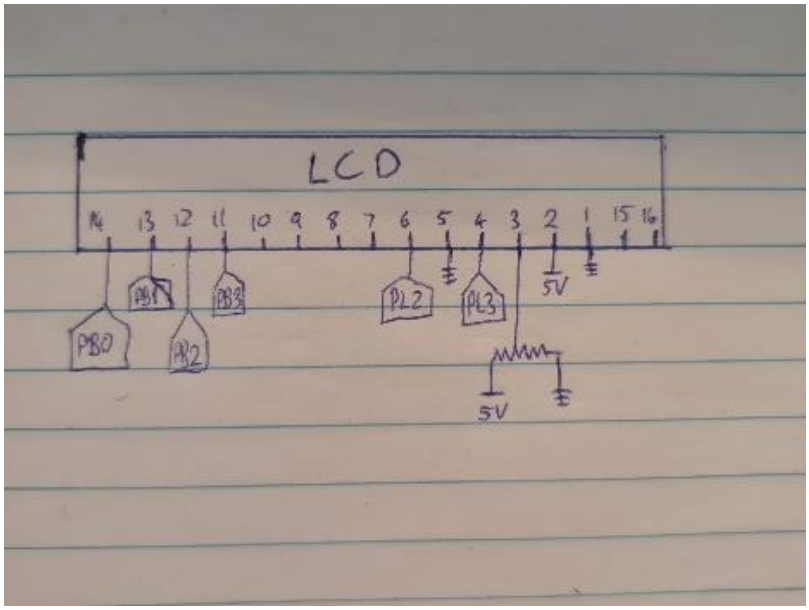


Figure 6: LCD Circuit Diagram

- The LCD is used to print out what mode the robot is in, autonomous or manual.

5.5 Push button

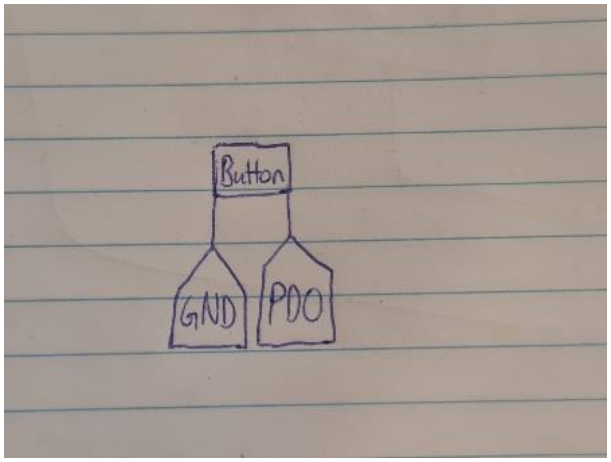


Figure 7: Push Button Circuit Diagram

- When pressed, the push button enters an interrupt service routine, which sees what state the robot is currently in (i.e.: autonomous or manual), and switches that state, by changing the value of a global variable, and printing to the LCD to tell the user.

5.6 Range sensors

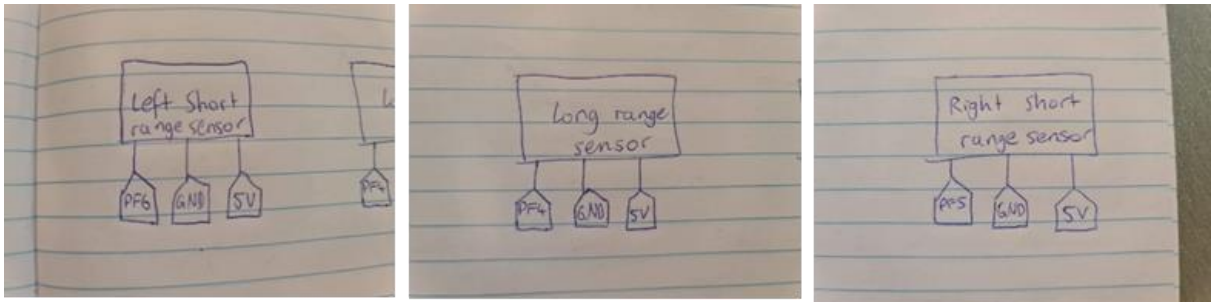


Figure 8: Range Sensor Circuit Diagrams

- In order to use the range sensors, they had to be calibrated, by measuring distances and taking note of the raw value that is returned. We then inverted that raw value, so that we could make a linear graph, which when plotted against the distance, gives us a formula for the conversion of the raw reading value, to the distance read by the range sensor. The formulae obtained are shown below.
  - $Short\ range\ sensor: y = (4 \times 10^{-5}x) + 0.0002$
  - $Long\ range\ sensor: y = (1 \times 10^{-5}x) + 0.0015$
- Using these formulas that we obtained, we created a function to find the distance (x) from the equation. These functions are shown below:
  - $Short\ range\ sensor: distance = \left(\frac{\frac{1}{raw\ reading} - 0.0002}{0.00004}\right)$
  - $Long\ range\ sensor: distance = \left(\frac{\left(\frac{1}{raw\ reading} - 0.0015\right)}{0.00001}\right)$

5.7 Servo motor

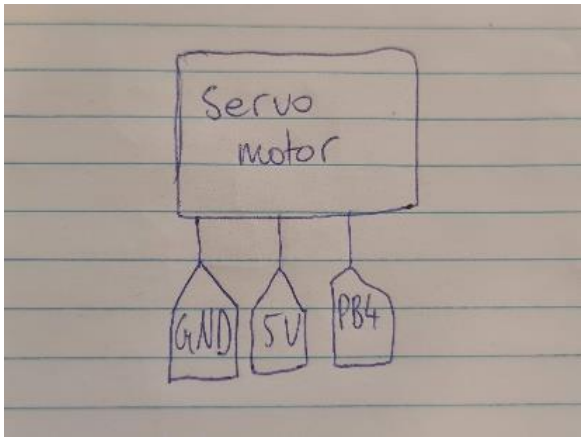
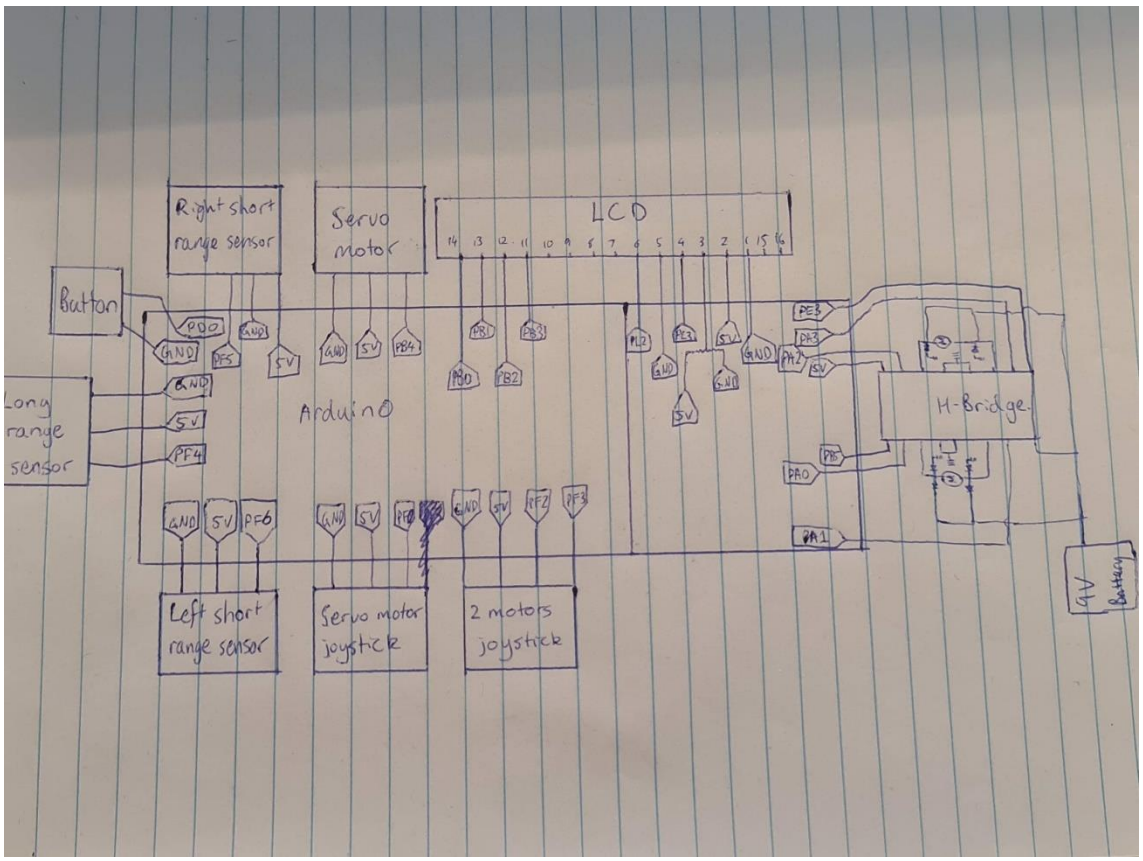


Figure 9: Servo Motor Circuit Diagram

- A position servo motor was used, which has a range of $-180°$ to $+180°$. Using the equation below, which uses a calculated top value at $50\ Hz$, we can change the position of the servo motor from a joystick.
  - $Top = \frac{16 \times 10^6\ Hz}{50 \times Hz \times 2 \times 8}$
  - $\therefore Top = 20\ 000$
  - $value = \frac{horizontal\ value}{1023} \times (2500 - 600) + 600$

6.0 Schematic diagram with description



7.0 Conclusion

- To conclude, we have successful built a semi-autonomous robot as per the technical demonstrations listed in the project briefing. In this report there contains a run down of all relevant calculations we have used to achieve this build. As per listed in this report is a schematic diagram of our overall build and circuit diagrams that represent individual circuit components and their respective pin connections. We have displayed the logic of our code via three flow charts, one for determining the push button state in order to determine which piece of code to run, one flow chart to show the logic behind manual mode and one to show the logic behind autonomous mode. For reference the code which mimics the flow charts can be found in appendix item 8.2. From our technical demonstrations as well as the aforementioned components of this report we have successfully achieved the technical demonstrations and therefore the project task.

8.0 Appendix

8.1 Picture of final build

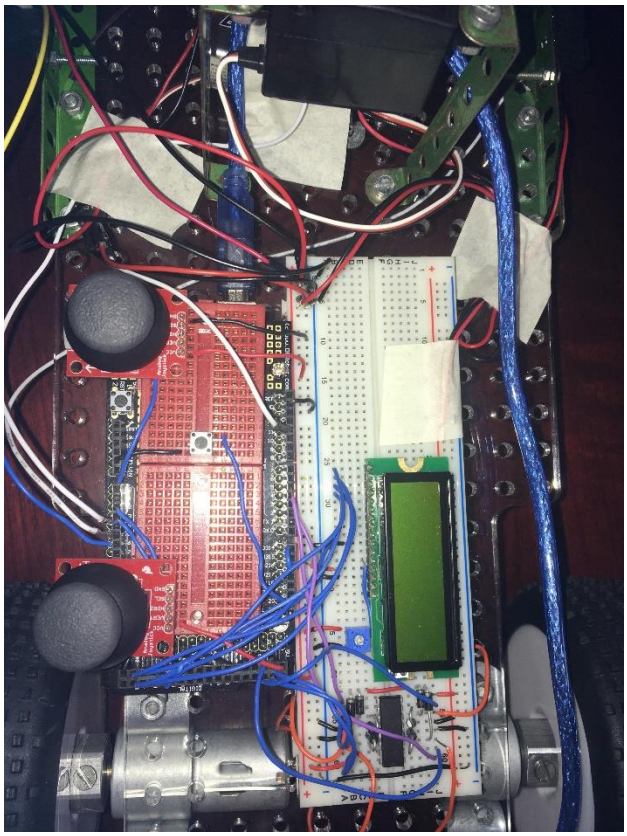


Figure 10: Finish Build Product Photo

8.2 Code

```
1.  //Author: Daniel Waits & Jason Kim
2.  //Created: 30/05/20
3.  //Final Project Code
4.
5.  #include "Controller.h"
6.
7.  static char serial_string[200] = {0};     // Declare and initialise string for serial printing
8.  static char lcd_string[33] = {0};         // Declare and initialise string for LCD Printing
9.
10.
11. uint32_t recvDataByte1=0, recvDataByte2=0, recvDataByte3=0, recvDataByte4=0,
    recvDataByte5 = 0, recvDataByte6 = 0, recvDataByte7 = 0;    // data bytes received
12. uint8_t serial_fsm_state=0;               // used in the serial receive ISR
13. bool new_message_received_flag=false;
14.
15.
16. // Push Button ISR for changing between autonomous and manual mode
17. int button = 0;
18. static const uint8_t debounceDelay = 20;
19.
20. ISR(INT1_vect) // When the push button is pressed
21. {
22.   if (button == 0)
```

```
23.  {
24.    lcd_home();
25.    lcd_clrscr();
26.    lcd_puts("Autonomous mode");
27.    button = 1;
28.  }
29.  else
30.  {
31.    lcd_home();
32.    lcd_clrscr();
33.    lcd_puts("Manual mode");
34.    button = 0;
35.  }
36.
37.  // Button debounce
38.  static uint32_t button1 = 0;
39.
40.  if(milliseconds > (button1 + debounceDelay))
41.  {
42.    button1 = !button1;
43.    TCCR1B ^= (1<<CS11); //Timer Counter 1 Control Register B and sets the prescaler to 0
44.    button1 = milliseconds;
45.  }
46.
47.  return;
48. }
49.
50. int main(void)
51. {
52.  //Initialisations
53.  adc_init();
54.  lcd_init();
55.  serial0_init();  // Terminal communication with PC
56.  serial3_init();  // microcontroller communication to/from another Arduino
57.  _delay_ms(20);
58.
59.  milliseconds_init();
60.
61.  //SERVO CONTROL MOTOR
62.  //Set OCR1B
63.  DDRB |= (1<<5);  //Enable PB5 for PWM - Servo Motor
64.
65.  //Initialise Motors
66.  TCCR3A = 0; TCCR3B = 0;
67.  TCCR3B |= (1<<WGM33);    //Turn PWM (Phase Correct) Mode On (MODE 8) for Timer 3
68.  TCCR3A |= (1<<COM3A1);   //Clear on up-count, set on down-count
69.  TCCR3B |= (1<<CS31);     //Sets prescalar to 8
70.  ICR3 = 199;             //TOP Value
71.
72.  //Initialise Servo Motor
73.  TCCR1A = 0;             //Timer Counter 1 Control Register A and disables hardware output
```

```
74.  TCCR1B = 0;            //Timer Counter 1 Control Register B and disables hardware output
75.  TCCR1B |= (1 << WGM13);   //Turn PWM (Phase Correct) Mode On (MODE 8)
76.  TCCR1A |= (1 << COM1A1);  //Toggles OC1A on Compare Match & OC1B and OC1C
     Disconnected (Setting the compare mode)
77.  TCCR1B |= (1<<CS11);      //Set Prescalar to 8
78.  ICR1 = 20000;           //TOP Value = (50 Hz * 2 * 8)/(16 x 10^6 Hz)
79.
80.  //Initialise button
81.  DDRD &= (1<<PD0);
82.  PORTD |= (1<<PD0);
83.  EICRA |= (1<<ISC01);
84.  EICRA &= ~(1<<ISC00);
85.  EIMSK |= (1<<INT1);
86.  sei();
87.
88.  //initialise range sensor
89.  double raw_reading_front;
90.  double raw_reading_left;
91.  double raw_reading_right;
92.  uint16_t mmValueShortLeft;
93.  uint16_t mmValueShortRight;
94.  uint16_t mmValueLong;
95.
96.  //Data Bytes to send
97.  uint32_t sendDataByte1=0, sendDataByte2=0, sendDataByte3=0, sendDataByte4=0,
     sendDataByte5 = 0, sendDataByte6 = 0, sendDataByte7 = 0;
98.  uint32_t current_ms=0, last_send_ms=0;      // Used for timing the serial send
99.  UCSR3B |= (1 << RXCIE3);                 // Enable the USART Receive Complete interrupt
     (USART_RXC)
100.
101.            while(1)
102.            {
103.              current_ms = milliseconds;
104.              //Sending section
105.              if(current_ms-last_send_ms >= 100) //sending rate controlled here one message
     every 100ms (10Hz)
106.              {
107.
108.                //Servo Joystick
109.                uint16_t horz_read = adc_read(PF0); //Reads the voltage at ADC0
110.                uint16_t duty1 = 0; //Servo Motor 1 Duty Value
111.                duty1 = (((double)horz_read/(double)1023)*(2500-600))+600;
112.                sendDataByte1 = ((double)duty1*(double)253)/((double)2500);
113.
114.
115.                //Manual Mode
116.               if (button == 0)
117.               {
118.
119.                //Motor
120.                uint32_t vertical = adc_read(PF2);
```

```
121.              uint32_t horizontal = adc_read(PF3);
122.              uint32_t horizontalRev = abs((horizontal-512)*199/512);
123.              uint32_t horizontalFor = (horizontal-512)*199/512;
124.              uint32_t verticalRev = abs((vertical-512)*199/512);
125.              uint32_t verticalFor = (vertical-512)*199/512;
126.
127.              sendDataByte2 = vertical/((double)2);
128.              sendDataByte3 = horizontal/((double)2);
129.              sendDataByte4 = horizontalRev;
130.              sendDataByte5 = horizontalFor;
131.              sendDataByte6 = verticalRev;
132.              sendDataByte7 = verticalFor;
133.
134.             serial0_print_string(serial_string);
135.             }
136.             //Autonomous mode
137.             else if (button == 1)
138.             {
139.              raw_reading_front  = adc_read(4);
140.              raw_reading_left = adc_read(6);
141.              raw_reading_right = adc_read(5);
142.              mmValueShortLeft = ((((1/raw_reading_left)-.0002))/.00004);
143.              if (mmValueShortLeft > 300)
144.              {
145.                mmValueShortLeft = 0;
146.              }
147.              mmValueShortRight = ((((1/raw_reading_right)-.0002))/.00004);
148.              if (mmValueShortRight > 300)
149.              {
150.               mmValueShortRight  = 0;
151.              }
152.              mmValueLong = ((((1/raw_reading_front)-.0015))/.00001);
153.              if (mmValueLong > 800)
154.              {
155.                mmValueLong = 0;
156.              }
157.
158.              if(mmValueLong > 200)
159.              {
160.               //go forwards
161.               PORTA |= (1<<PA1);
162.               PORTA &= ~(1<<PA0);
163.               OCR3B = 199;
164.               PORTA |= (1<<PA3);
165.               PORTA &= ~(1<<PA2);
166.               OCR3A = 199;
167.              }
168.              if((mmValueLong < 200) && (mmValueShortLeft > mmValueShortRight)) //hits -|
    corner
169.              {
170.               //left 90 deg
```

```
171.                PORTA |= (1<<PA3);
172.                PORTA &= ~(1<<PA2);
173.                OCR3A = 199;
174.                PORTA |= (1<<PA0);
175.                PORTA &= ~(1<<PA1);
176.                OCR3B = 199;
177.               }
178.               if((mmValueLong < 200) && (mmValueShortLeft < mmValueShortRight)) //hits |-
    corner
179.               {
180.                //right 90 degrees
181.                PORTA |= (1<<PA2);
182.                PORTA &= ~(1<<PA3);
183.                OCR3A = 199;
184.                PORTA |= (1<<PA1);
185.                PORTA &= ~(1<<PA0);
186.                OCR3B = 199;
187.               }
188.              }
189.
190.             sprintf(serial_string, "left: %4d mm \t right: %4d mm \t front: %4d mm \n",
    mmValueShortLeft, mmValueShortRight, mmValueLong);
191.             serial0_print_string(serial_string);
192.
193.
194.              if (sendDataByte1>253)
195.              sendDataByte1 = 0;
196.              if (sendDataByte2>253)
197.              sendDataByte2 = 0;
198.              if (sendDataByte3>253)
199.              sendDataByte3 = 0;
200.              if (sendDataByte4>253)
201.              sendDataByte4 = 0;
202.              if (sendDataByte5>253)
203.              sendDataByte5 = 0;
204.              if (sendDataByte6>253)
205.              sendDataByte6 = 0;
206.              if (sendDataByte7>253)
207.              sendDataByte7 = 0;
208.
209.
210.              last_send_ms = current_ms;
211.              serial3_write_byte(0xFF);    //send start byte = 255
212.              serial3_write_byte(sendDataByte1);  //send first data byte: must be scaled to the
    range 0-253
213.              serial3_write_byte(sendDataByte2);  //send second parameter: must be scaled
    to the range 0-253
214.              serial3_write_byte(sendDataByte3);  //send first data byte: must be scaled to the
    range 0-253
215.              serial3_write_byte(sendDataByte4);  //send second parameter: must be scaled
    to the range 0-253
```

```
216.              serial3_write_byte(sendDataByte5);  //send second parameter: must be scaled
    to the range 0-253
217.              serial3_write_byte(sendDataByte6);  //send second parameter: must be scaled
    to the range 0-253
218.              serial3_write_byte(sendDataByte7);  //send second parameter: must be scaled
    to the range 0-253
219.              serial3_write_byte(0xFE);     //send stop byte = 254
220.             }
221.
222.             //if a new byte has been received
223.             if(new_message_received_flag)
224.             {
225.               // now that a full message has been received, we can process the whole message
226.               // the code in this section will implement the result of your message
227.               // sprintf(serial_string, "servo:%4d motor forward:%4d backward:%4d left:%4d
    right: %4d\n", recvDataByte2, recvDataByte3, recvDataByte4, recvDataByte5,
    recvDataByte6);
228.               //serial0_print_string(serial_string);  // print the received bytes to the USB serial
    to make sure the right messages are received
229.
230.               new_message_received_flag=false;  // set the flag back to false
231.             }
232.            }
233.            return(1);
234.           } //end main
235.
236.
237.           ISR(USART3_RX_vect)  // ISR executed whenever a new byte is available in the serial
    buffer
238.           {
239.             uint8_t serial_byte_in = UDR3; //move serial byte into variable
240.
241.             switch(serial_fsm_state) //switch by the current state
242.             {
243.               case 0:
244.               //do nothing, if check after switch case will find start byte and set serial_fsm_state
    to 1
245.               break;
246.               case 1: //Servo Motor Control
247.               recvDataByte1 = ((double)serial_byte_in*(double)2500)/((double)253);
248.               OCR1A = (recvDataByte1);
249.               serial_fsm_state++;
250.               break;
251.               case 2: //waiting horizontal variable
252.               recvDataByte2 = serial_byte_in *((double)2);
253.               serial_fsm_state++;
254.               break;
255.               case 3: //waiting vertical variable
256.               recvDataByte3 = serial_byte_in* ((double)2);
257.               serial_fsm_state++;
258.               break;
```

```
259.            case 4: //waiting for horizontalRev variable
260.            recvDataByte4 = serial_byte_in;
261.            serial_fsm_state++;
262.            break;
263.            case 5: //waiting for horizontalFor variable
264.            recvDataByte5 = serial_byte_in;
265.            serial_fsm_state++;
266.            break;
267.            case 6: //waiting for verticalRev variable
268.            recvDataByte6 = serial_byte_in;
269.            serial_fsm_state++;
270.            break;
271.            case 7: //waiting for verticalFor variable
272.            recvDataByte7 = serial_byte_in;
273.             //Not moving
274.               if(recvDataByte2 > 490 && recvDataByte2 < 530)
275.               {
276.                 PORTA |= (1<<PA0);
277.                 PORTA |= (1<<PA1);
278.               }
279.
280.               if(recvDataByte3 > 490 && recvDataByte3 < 530)
281.               {
282.                 PORTA |= (1<<PA2);
283.                 PORTA |= (1<<PA3);
284.               }
285.               //Backwards
286.               if(recvDataByte2 > 530)
287.               {
288.                 PORTA |= (1<<PA0);
289.                 PORTA &= ~(1<<PA1);
290.                 OCR3B = recvDataByte5;
291.                 PORTA |= (1<<PA2);
292.                 PORTA &= ~(1<<PA3);
293.                 OCR3A = recvDataByte5;
294.               }
295.               //Forwards
296.               if(recvDataByte2 < 490)
297.               {
298.                 PORTA |= (1<<PA1);
299.                 PORTA &= ~(1<<PA0);
300.                 OCR3B = recvDataByte4;
301.                 PORTA |= (1<<PA3);
302.                 PORTA &= ~(1<<PA2);
303.                 OCR3A = recvDataByte4;
304.               }
305.               //Right
306.               if(recvDataByte3 < 490)
307.               {
308.                 PORTA |= (1<<PA2);
309.                 PORTA &= ~(1<<PA3);
```

```
310.                OCR3A = recvDataByte7;
311.                PORTA |= (1<<PA1);
312.                PORTA &= ~(1<<PA0);
313.                OCR3B = recvDataByte7;
314.              }
315.              //Left
316.              if(recvDataByte3 > 530)
317.              {
318.                PORTA |= (1<<PA3);
319.                PORTA &= ~(1<<PA2);
320.                OCR3A = recvDataByte6;
321.                PORTA |= (1<<PA0);
322.                PORTA &= ~(1<<PA1);
323.                OCR3B = recvDataByte6;
324.              }
325.            break;
326.            case 8: //waiting for stop byte
327.            if(serial_byte_in == 0xFE) //stop byte
328.            {
329.              // now that the stop byte has been received, set a flag so that the
330.              // main loop can execute the results of the message
331.              new_message_received_flag=true;
332.            }
333.            // if the stop byte is not received, there is an error, so no commands are
    implemented
334.            serial_fsm_state = 0; //do nothing next time except check for start byte (below)
335.            break;
336.          }
337.          if(serial_byte_in == 0xFF) //if start byte is received, we go back to expecting the
    first data byte
338.          {
339.            serial_fsm_state=1;
340.          }
341.        }
```