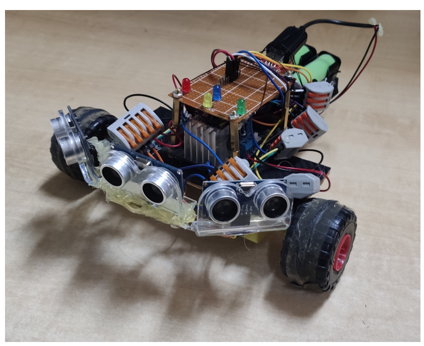
MarsCar

Lecturer:
Maoz Loants

Lab assistant:
Adam Sofer

Vitali Lopushenko	Osher saragani	Uriel Almassy
317104347	315348706	305459372

22.01.2020



Contents

1 Requirements specification

The Requirements is an functional small size vehicle that can drive without any intervention from the user and avoid any kind of obstacle.

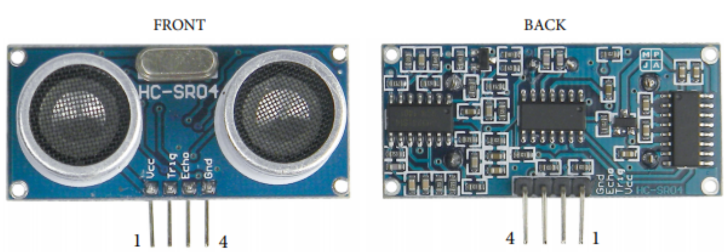
2 System specification

2.1 Parts

2.1.1 Ultrasonic sensor

An ultrasonic sensor is an instrument that measures the distance to an object using ultrasonic sound waves. An ultrasonic sensor uses a transducer to send and receive ultrasonic pulses that relay back information about an object's proximity. High-frequency sound waves reflect from boundaries to produce distinct echo patterns. The working principle of this module is simple. It sends an ultrasonic pulse out at 40kHz which travels through the air and if there is an obstacle or object, it will bounce back to the sensor. By calculating the travel time and the speed of sound, the distance can be calculated.

$Distance = \frac{Time * SpeedOfSound}{2}$

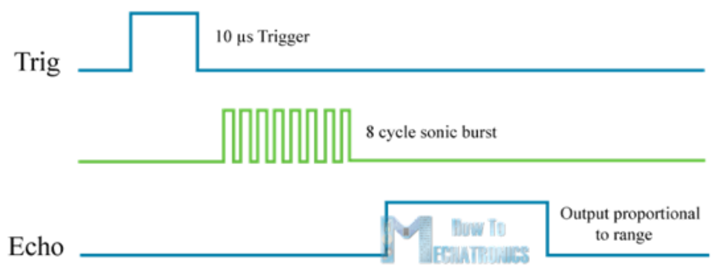


Pinout:

1. Vcc - 5V
2. Trig
3. Echo
4. Ground

Operating voltage: 5V
Operating Current: 15mA
Measure Angle: 15
Ranging Distance: 2cm - 4m

To generate sound waves keep Trig High for 10 microseconds. These waves will travel at the speed of sound, creating 8 cycle sonic burst that will be collected in the Echo pin. The echo pin remains turned on for the time these waves take to travel and bounce back to the receiving end. This sensor is mainly incorporated with Arduino to measure the required distance.



2.1.2 DC motor

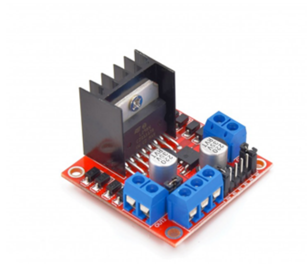
DC motors are the most common type of motors used in robotics. DC motors appear in a large variety of shapes and sizes. DC motors have mostly two terminals, across the voltage is applied. When the voltage is applied across these terminals, the motor starts to spin in one direction, and when the polarity of applied voltage is reversed the direction of the rotation is also reversed.
In our Project we used simple geared DC motor

Specification

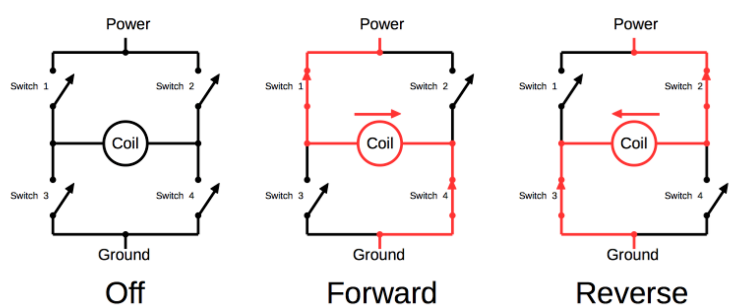
1. Operating voltage: 3-12V DC
2. The load current: 70 mA (3 V) (250 mA MAX)
3. Maximum torque: 800gf
4. Turning speed: 1:48

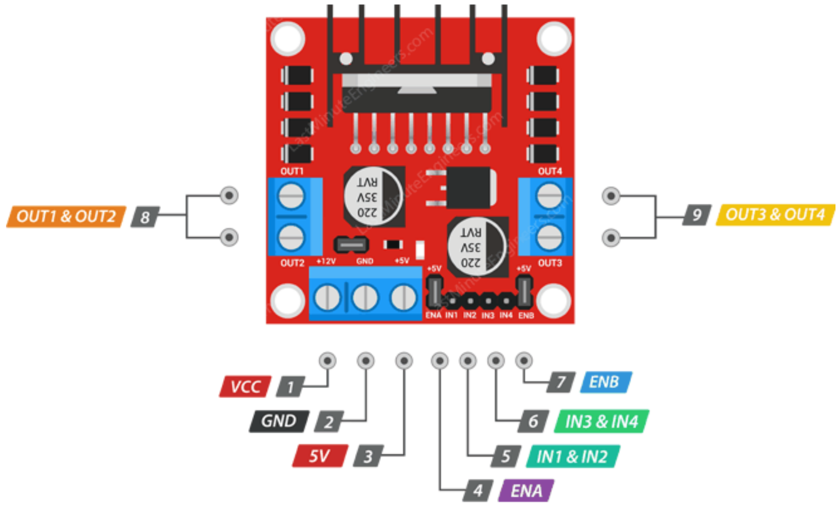
2.1.3 H-bridge

An H bridge is an electronic circuit that switches the polarity of a voltage applied to a load. These circuits are often used in robotics and other applications to allow DC motors to run forwards or backwards.
we used L298N H-bridge.



we can see four switches which are all in the open or “off” position. In the center of the circuit is a DC motor. If you look at the circuit as it is drawn here you can distinctly see a letter “H”, with the motor attached in the center or bridge section – thus the term H-Bridg”.





Pinout:

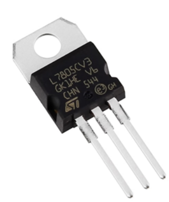
1. Vcc: 5-24V DC
2. Ground
3. 5V for the circuit
4. Left motor PMW enable
5. Left motor input
6. Right motor input
7. Right motor PMW enable
8. Left motor connectors
9. Right motor connectors

2.1.4 Power suplly

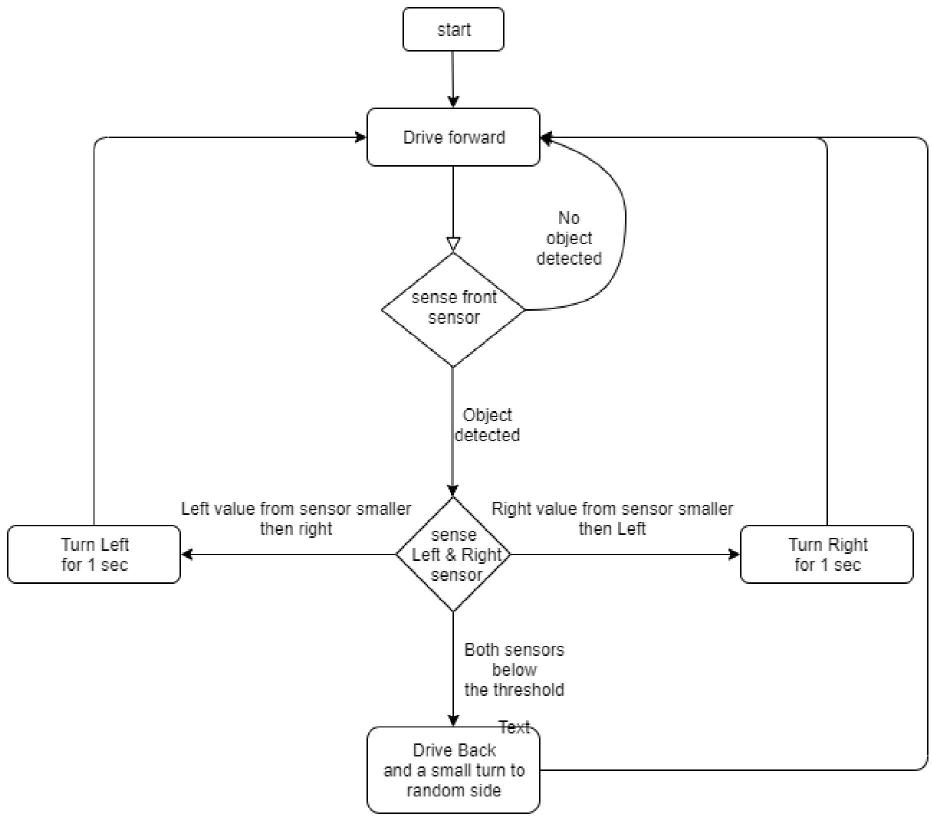
We used 2 18650 batteries:

1. Capacity: 3400mah
2. Voltage 3.2-4.2 (Full-Empty)

To achieve the desired operating voltage we connected 2 cells in series to achieve at least 6.4V. We also used 5V voltage regulator to supply stable voltage.



2.2 Block diagram



2.3 Usage and limitation

To use the vehicle you need to turn on the system by pressing the on button. system limitation: not water proof, do not operate under 0 Celsius degree or above 40 and can travel only on flat surface.
Because the following limitation DO NOT USE IT ON MARS.

3 Algorithm

The algorithm is very simple and described in the block diagram.

1. Drive forward
2. constantly sense the front sensor, if an object detected 5 sensing in a row(Just to be sure that is an object and not false reading)
3. sense Left and Right sensor: if left sensor value less then the right go to 4, if right smaller go to 5, if both sensor reading below threshold go to 6
4. Turn left for 1 sec: go to 1
5. Turn Right for 1 sec: go to 1
6. Drive back and turn to one of the sides a bit: go to 1

4 Optimization

in our project there is not much we can do in power consumption because the vehicle always need to sense if there is object in front of it and drive forward, but we tried to optimize the pins we used, each ultraSonic sensor has 2 pins for data (Trig and Echo) instead using 6 pins for 3 sensor we used only 3 pins by changing the direction of the pin, before sending trig we define the pin as output, after sending 10 micro seconds signal we switched the direction to input for receiving the Echo signal.

5 Problems

Most of our problems were hardware problems, list of problems:

1. we tried to optimize the pin use our idea was to connect all the echo pins together to reduce pins we used but the value we received were not right.
2. while taking measurements sometimes the value that we received were incorrect, so the decision if there is an object in front of us is only if 5 samples in a row bellow the threshold.

3. while debugging we didnt use the same ground for the MSP and the sensor, that causing an strange behavior.
4. Defected sensors
5. Compiler optimization cause strange behavior.
6. Loose connection make a lot of problems so we use soldering iron to make good connections.

6 Power consumption

After booting up the system by turning it on, the vehicle runs at full power.

1. DC Motor: 250 $mA(*2)$
2. Sensors: 15 mA (always measuring, one at time)
3. H bridge (Logical circuit): 36 mA
4. MSP430: 270 μA
5. LED: 30 mA $(*4)$

7 Memory and Functions

	Main	Detect	interrupt timer	Interrupt port 2
מה הפונקציה עושה	מריצה את האלגוריתם בזמן הנסיעה	שולחת ultra-sonic pulse לכיוון direction	מקדמת את ערך ספירת הזמן במשתנה הגלובלי milliseconds	בודקת אם התקבל הגל חזרה אם כן מחשבת כמה זמן עבר משליחתו
פרמטרים	אין	Direction – ערך ייחודי לכל כיוון המציין את הביט דרכו אנו מקבלים מידע מהחיישן בפורט 2	אין	אין
ערך החזרה	אין	ממירה את הערך הנמדד מהחיישן למרחק ומחזירה אותו בסנטימטרים	משנה את ערך המשתנה הגלובלי milliseconds – כמה זמן עבר משליחת הpulse	משנה את ערך המשתנה הגלובלי sensor לערך הנמדד מהחיישן
גודל	398B	88B	4B	82B

Total memory usage: Flash:1012 B, Stack:1224 B

8 The code:

```
#include <msp430.h>
//#define RECEIVE_ECHO 0x01 //[P3,P2.0]Receive echo from ultra-sonic sensor
#define FORWARD 0x02      //[P4,P2.1]Trigger forward wave
#define LEFT 0x04         //[P5,P2.2]Trigger left wave
#define RIGHT 0x08        //[P6,P2.3]Trigger right wave
#define MOTOR_LF   0x10   //[P7,P2.4]Activate left motor - forward Yellow
#define MOTOR_RF   0x08   //[P8,P4.3]Activate right motor - forward Red
#define MOTOR_LR   0x10   //[P9,P4.4]Activate left motor - reverse Green
#define MOTOR_RR   0x20   //[P10,P4.5]Activate right motor - reverse BLUE
#define READ_PERIOD 5     //Time to wait between trigger wave and receive echo
#define TIMEOUT 30000     //Maximum time to wait if no ehco is received
#define DRIVE_TIME 1000000
#define MIN_DIS 20        //The minimal distance in which the vehicle can respond
#define AMOUNT_SAMPLE 5   //The number of pulses the ultra-sonic sensor transmit in order to detect blockage
#define MIN_DIS_TURN 5    //The minimal distance in which the vehicle can make a turn
#define CONVERSION_CONST 58

int currentEcho = 0;
int miliseconds = 0;      //A counter to track how many milliseconds have passed
int sensor = 0;           //value from ultra-sonic sensor

int detect(int direction);  //return the distance in cm from the next blockage, the parameter 'trigger' represent which sensor to activate
```

```
void main(void)
{
  BCSCTL1 = CALBC1_1MHZ;
  DCOCTL = CALDCO_1MHZ;                         // submainclock 1mhz
  WDTCTL = WDTPW + WDTHOLD;                     // Stop WDT

  CCTL0 = CCIE;                                 // CCR0 interrupt enabled
  CCR0 = 1000;                                  // 1ms at 1 mhz
  TACTL = TASSEL_2 + MC_1;                      // SMCLK, upmode

  P2IFG  = 0x00;                                //clear all interrupt flags
  P1IFG  = 0x00;                                //clear all interrupt flags
  P1DIR |= 0x03;                                // P1.0 P1.1 as output for LEDs

  P4DIR |= MOTOR_RR + MOTOR_RF + MOTOR_LR; // set pins' direction  as output
  P2DIR |= MOTOR_LF;                            //set direction pin as output


  _BIS_SR(GIE);                                 // global interrupt enable

  int left_average = 0,right_average = 0,i;

  P4OUT &= ~MOTOR_RF;
  P4OUT &= ~MOTOR_RR;
  P2OUT &= ~MOTOR_LF;
  P4OUT &= ~MOTOR_LR;
```

```
  int samplesInRow = 0;
  while(1)
    {
      P1OUT &= ~0x03;
      P4OUT &= ~MOTOR_RF;
      P4OUT &= ~MOTOR_RR;
      P2OUT &= ~MOTOR_LF;
      P4OUT &= ~MOTOR_LR;

    //activate motors - start moving ahead!
    P4OUT |= MOTOR_RF;
    P2OUT |= MOTOR_LF;
    //keep moving & detect forward as long as there is no blockage
    samplesInRow = 0;
    while(samplesInRow < AMOUNT_SAMPLE)
    {
        if(detect(FORWARD) < MIN_DIS)
            samplesInRow++;
        else
            samplesInRow = 0;
    }

    //stop moving forword in order to make a turn
    P2OUT &= ~MOTOR_LF;
    P4OUT &= ~MOTOR_RF;
    samplesInRow = 0;

    right_average = 0;
    left_average = 0;

    //gather samples from the sides
    for(i=0; i<AMOUNT_SAMPLE;i++)
    {
        //left
        left_average += detect(LEFT);
        //right
        right_average += detect(RIGHT);
    }

```

```c
    //calc average for improved accuracy
    right_average /= AMOUNT_SAMPLE;
    left_average  /= AMOUNT_SAMPLE;


    //__delay_cycles(100000);
    //if a turn is not an option goto reverse
    if(right_average<MIN_DIS_TURN  && left_average<
  MIN_DIS_TURN)
    {
        //reverse
        P4OUT |= MOTOR_LR;
        P4OUT |= MOTOR_RR;
        __delay_cycles(DRIVE_TIME);
        continue;
    }
    if(right_average < left_average)
    {
     //turn left
        //activate left motor - forward
        P2OUT |= MOTOR_LF;              // activate
  motor to go faorward
        //activate right motor - reverse
        P4OUT |= MOTOR_RR;              // activate
  motor to go faorward
        P1OUT |= 0x01;
    }
    else
    {
     //turn right
        //activate left motor - reverse
        P4OUT |= MOTOR_LR;              // activate
  motor to go faorward
        //activate right motor - forward
        P4OUT |= MOTOR_RF;              // activate
  motor to go faorward
        P1OUT |= 0x02;
    }
    __delay_cycles(DRIVE_TIME);
  }//while(1)
```

```
}//main

#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{
    if( P2IFG & currentEcho )           //is there interrupt pending?
        {
          if(!( P2IES & currentEcho ))  // is this the rising edge?
          {
            TACTL|=TACLR;               // clears timer A
            miliseconds = 0;
            P2IES |= currentEcho;       //falling edge
          }
          else
          {
              sensor = (long)miliseconds*1000 + (long)TAR; //calculating RECEIVE_ECHO lenght
          }
    P2IFG &= ~currentEcho;        //clear flag
    }
    P2IFG = 0x00;
}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
  miliseconds++;
}
int detect(int trigger)
{
    currentEcho = trigger;
    P2IE &= ~currentEcho;               // disable interrupt
    P2DIR |= trigger;                   // make pin P2.0 output (trigger)[p3]
    P2OUT |= trigger;                   // generate pulse
    __delay_cycles(READ_PERIOD);        // for 10us
```

```
    P2OUT &= ~trigger;                     // stop pulse
    P2DIR &= ~currentEcho;                 // make pin P2.1 input (RECEIVE_ECHO)[p4]
    P2IFG = 0x00;                          // clear flag just in case anything happened before
    P2IE |= currentEcho;                   // enable interrupt on RECEIVE_ECHO pin
    P2IES &= ~currentEcho;                 // rising edge on RECEIVE_ECHO pin
    __delay_cycles(TIMEOUT);               // delay for 30ms (after this time echo times out if there is no object detected)
    currentEcho = 0;
    return sensor / CONVERSION_CONST;  // converting RECEIVE_ECHO lenght into cm
}
```