Project Report-EE4735

Hallway Navi Bot

Team Members: Kirk D'Souza

Mahesh Sarode

Guided by: Prof. Cischke

T.A. : Caoyang Jiang

Table of Contents

Sr No.	Contents	Page no.
1	Abstract and Objective	3
2	Design	4
3	Construction Layout	4
4	Pin Diagram	6
5	Program Description	7
6	Algorithm Flowchart	8
7	Algorithm Description	9
8	Hardware Used	10
9	Ultrasonic Sensors	10
10	Sabertooth 2x10	1
11	Applications	13
12	References	13
13	Appendix (with code)	14

Abstract and Objective:

An obstacle avoidance robot is an autonomous vehicular device that detects vertical surfaces and avoid collision with them via an-inbuilt program. A navigation bot is an autonomous vehicular device that can navigate it's way around a variable pathway. The objective is to make a robot able to navigate in a U-shaped hallway. The robot should avoid obstacles in its path and should not bump into walls and obstacles. It should stop within a meter at the other end of the hallway at 8th floor in EERC building.

Design:

Construction layout:

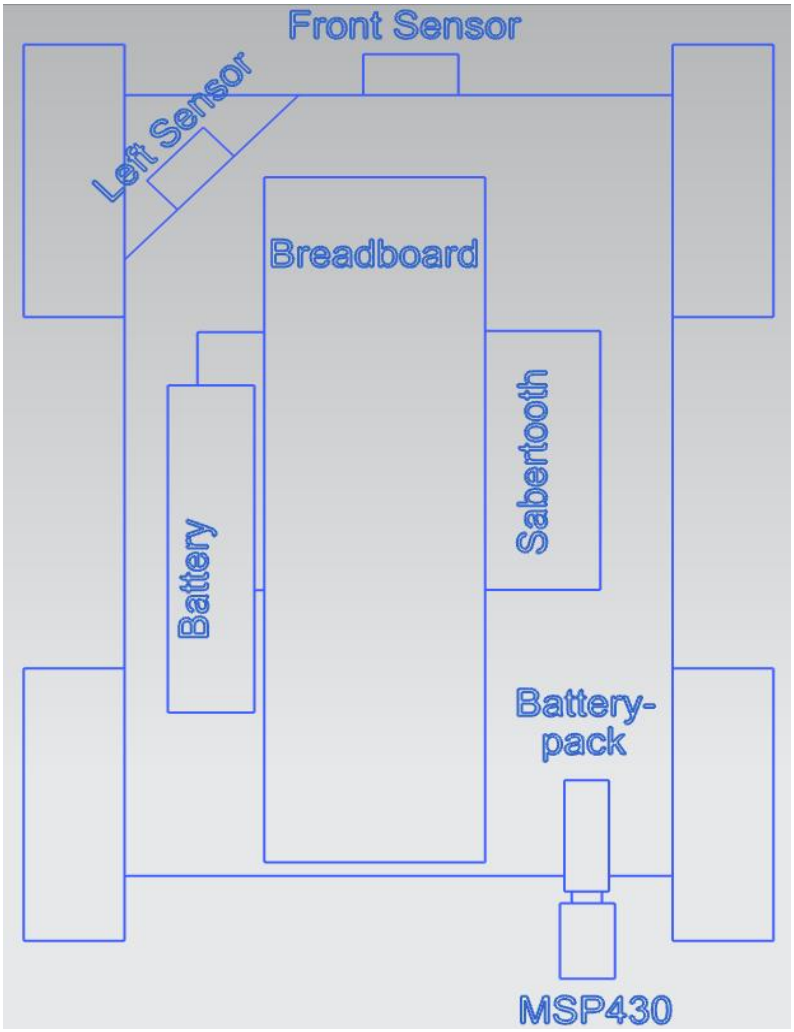


Figure 1.1 Robot layout with components

Ultrasonic sensors send sonic bursts and from echo generate a pulse width proportional to the distance of the object ahead. Two ultrasonic sensors are mounted. One for detecting front distance and other for detecting left distance. The left distance sensor is mounted at 45 degree as shown in figure 1. Using these sensors, the robot is able to navigate and avoid obstacles in its path. MSP430 has two timers with three channels each. Timer A channel 0 & channel 1 are used for reading the outputs from ultrasonic sensors. Timer B channel 1 is used to generate a trigger pulse which drives the ultrasonic sensors to send sonic bursts. Sabertooth motor controller is used in *Simplified Serial Mode* with baud rate of 9600. 11.1 V 3S battery Li-Po battery is used to power motor and voltage translator.

DIP switch configuration: 1-Up, 2-Down, 3-Down, 4-Down, 5-Up, 6-Up

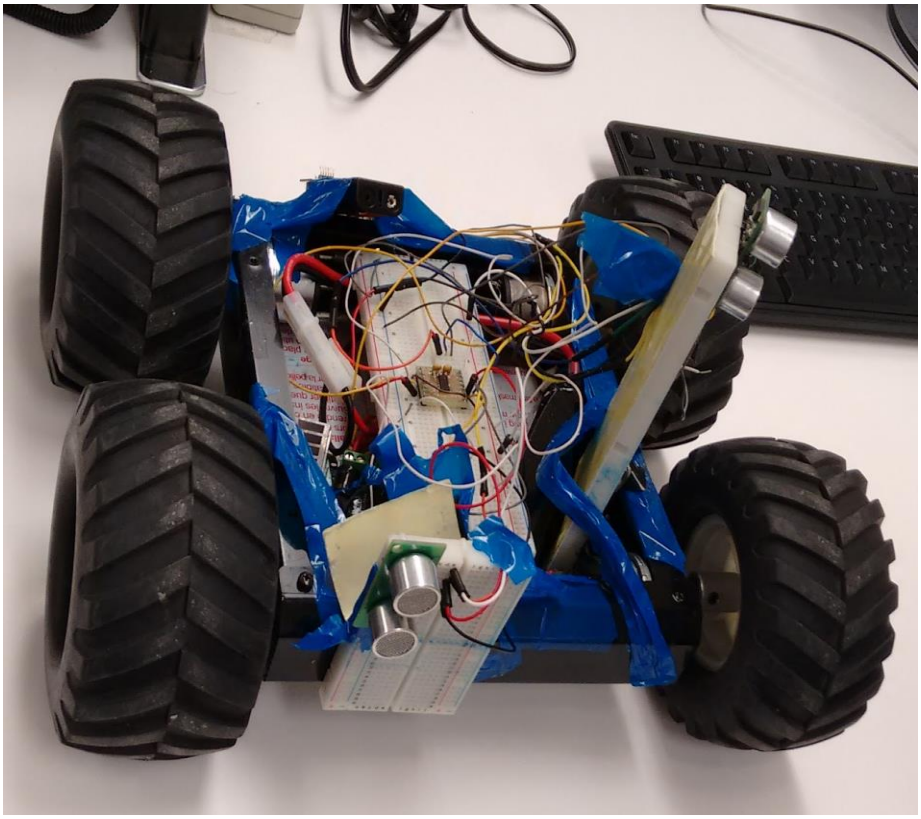


Figure 1.2 Hallway navi bot

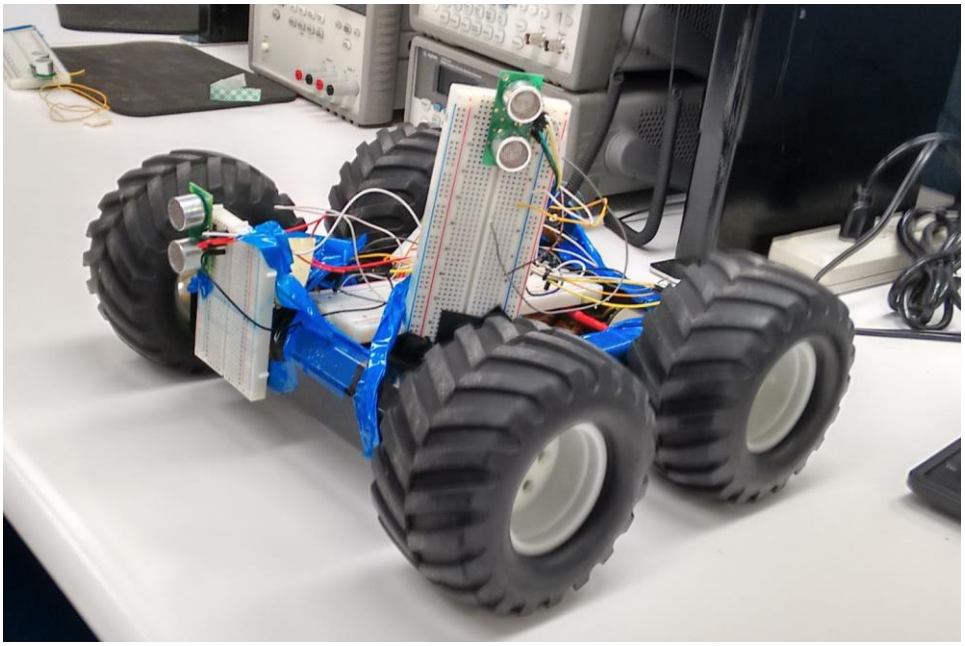


Figure 1.3 Hallway navi bot

Pin Diagram

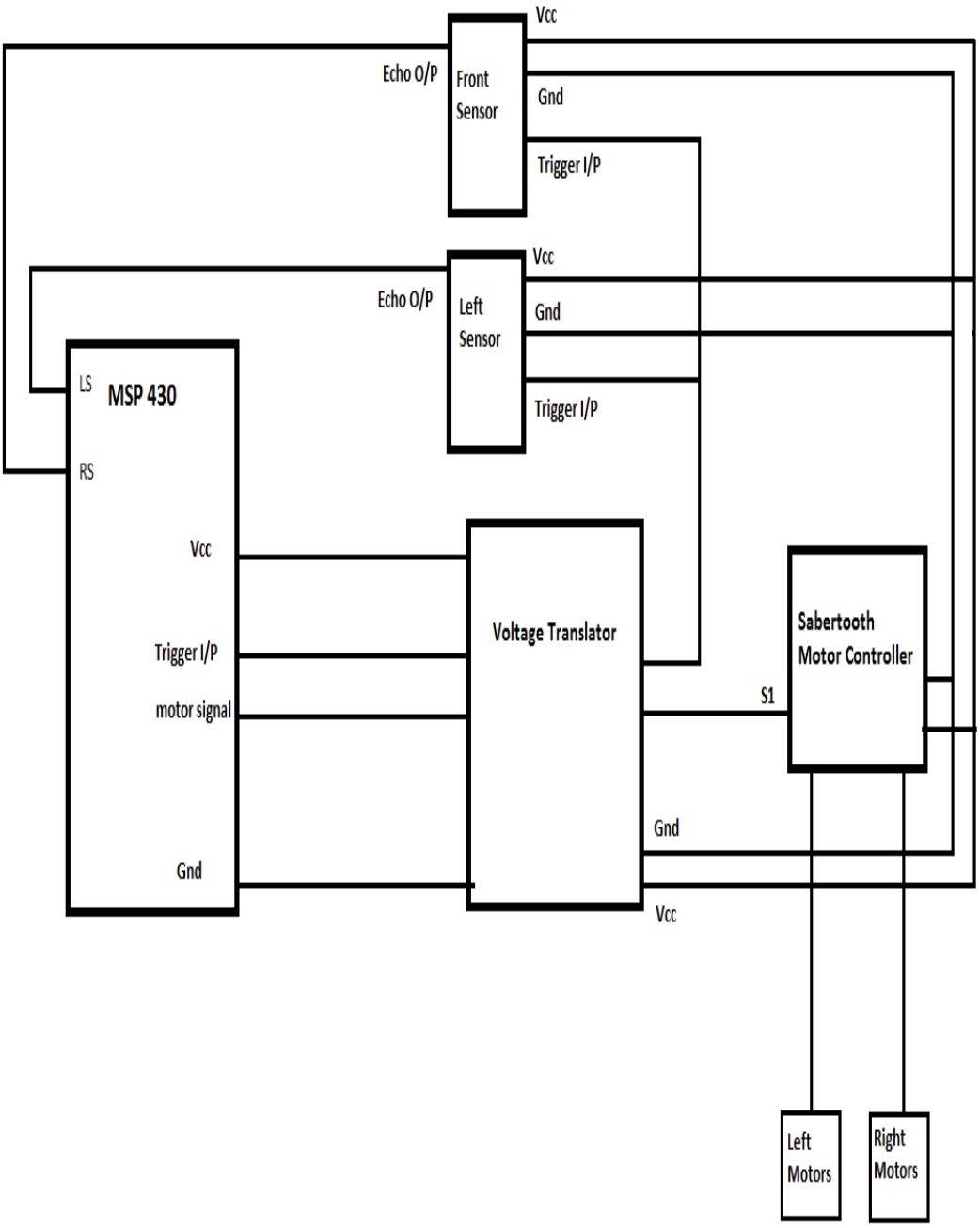


Figure 2. Pin diagrams for the Hallway Navi Bot.

Program Description:

Our bot is designed to avoid obstacles while navigating its' route in the hallway. The program is designed such that the bot will move forward in case of no obstacle while trying to maintain a distance from the left wall.
The width of the hallway is divided into 4 zones, and there are also 3 additional conditions that are defined as zones:

Hallway zones:

Zone 1: 25 to 100 cm from the left wall
Zone 2: 100 to 125 cm from the left wall
Zone 3: 125 to 150 cm from the left wall
Zone 4: 150 to 200 cm from the left wall

Additional Zones:

Zone 0: When bot is too close to obstacle
Zone 5: When a new hallway is detected on the left of the bot i.e. when the left sensor value exceeds 200 cm
Zone 6: When the robot is too close to a wall

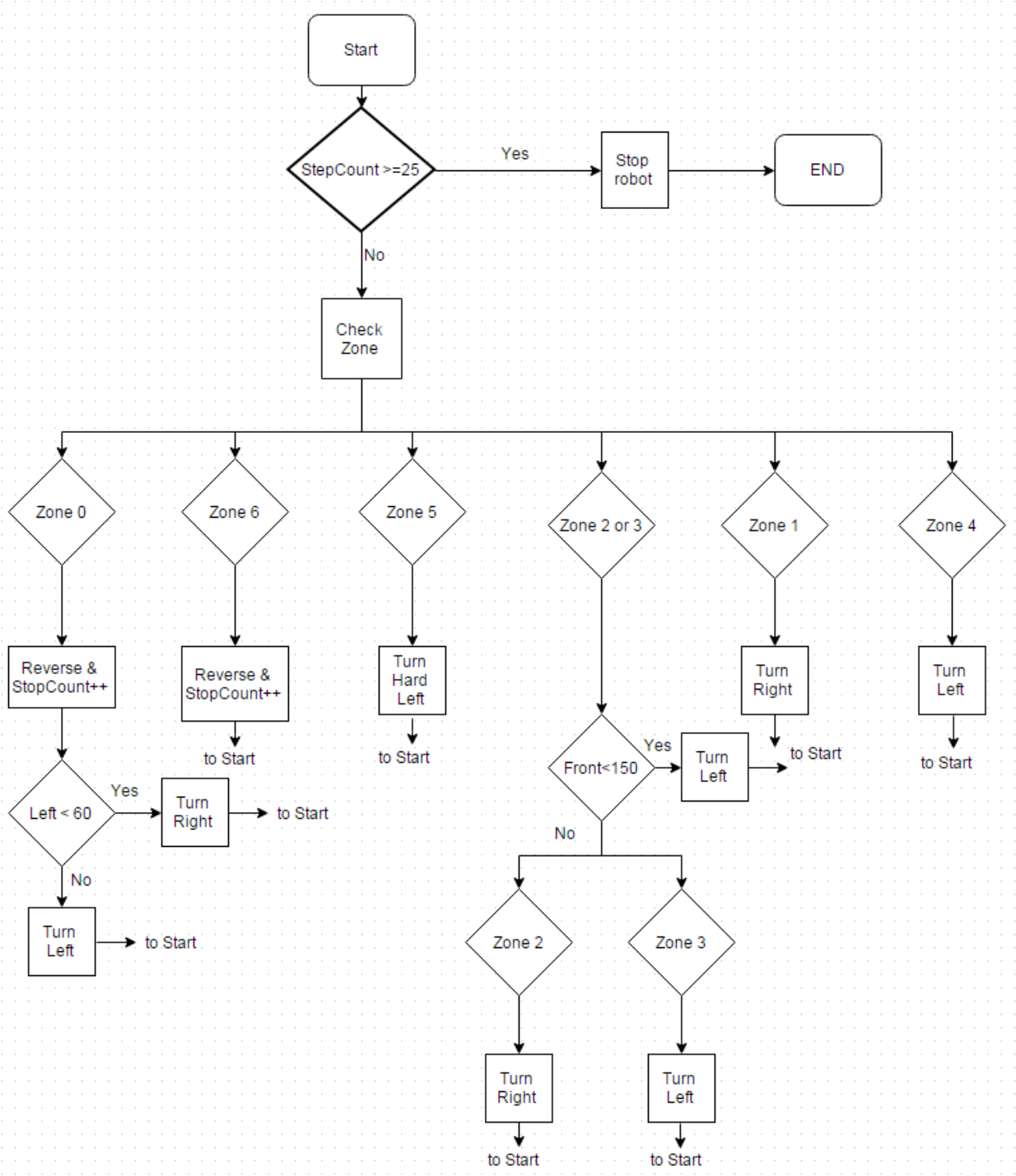
With zone 2 and 3 being the central zones of the hallway, the bot is to try and maintain itself between these zones.

If an obstacle is detected the bot will try to avoid it by swerving around it while still moving forward, similar to a car on a highway. If the obstacle/wall is too close (zone 0) the bot will reverse and seek to correct its' course accordingly.

The bot turns left towards the new hallway when there is a drastic increase in the value of the left sensor (zone 5).

When against a wall the bot will try to readjust itself by reversing several times in order to avoid collision with the wall. A stop counter is used to count the number of times the bot enters the condition with the reverse commands. When a certain limit is reached the bot stops in place.

Algorithm Flowchart:



Algorithm Description:

1) Scan the area with the front and left sensors and determine the zone and conditions the bot currently is in
2) Check Stop counter
3) If stop count is met, stop bot in place and end program
4) If not reached, proceed to step 5
5) If in Zone 0(i.e. too close to obstacle), then reverse bot and check left sensor value to determine which wall is closer
6) If closer to left wall, then reverse turn toward the right wall, and increment stop count
7) If closer to right wall, then reverse turn toward the left wall, and increment stop count
8) If in Zone 5, then spin left (anti clockwise), and reset the stop count
9) If in Zone 6, then reverse, and increment stop count
10) If in Zone 1, then gradually move forward towards Zone 2, if previous zone was Zone 0, then increment stop count
11) If in Zone 4, then gradually move forward towards Zone 3, and reset the stop count
12) Check if obstacle is detected in Zones 2 or 3
13) If detected then go left
14) If not detected then check bot's current zone
15) if in Zone 2, then gradually move forward towards Zone 3, if previous zone was Zone 0, then increment stop count
16) if in Zone 3, then gradually move forward towards Zone 2, and reset the stop count
17) End

Hardware Used:

Sr No.	Device	Quantity	Description
1	eZ430-RF2500	1	MSP430F2274 microcontroller chip
2	SRF 04	2	Ultrasonic sensors
3	Sabertooth 2x10	1	Motor controller
4	TXS0104E	1	Voltage translator

Table 1 - Hardware

Besides these devices, a Li-Po 11.1 V battery, breadboards, connectors and battery pack for MSP430, robot chassis with motors were used.

SRF-04 Ultrasonic sensors:

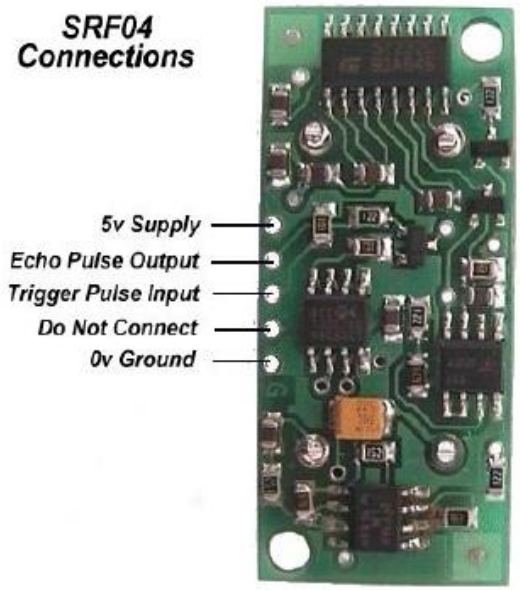


Figure 3. Pin connection for SRF-04 Ultrasonic sensor

The sensors need minimum 10 microseconds wide trigger pulse to operate. Current consumption is 30-50 mA. Frequency is 40 kHz and range of the sensor is 3 cm to 300 cm. Output is PWM signal with pulse width proportional to the distance of the object in front of the sensor. The trigger pulse, sonic burst and output pulse from the sensor can be seen in figure 4. Figure 1 shows the pin connections on SRF-04 sensor.

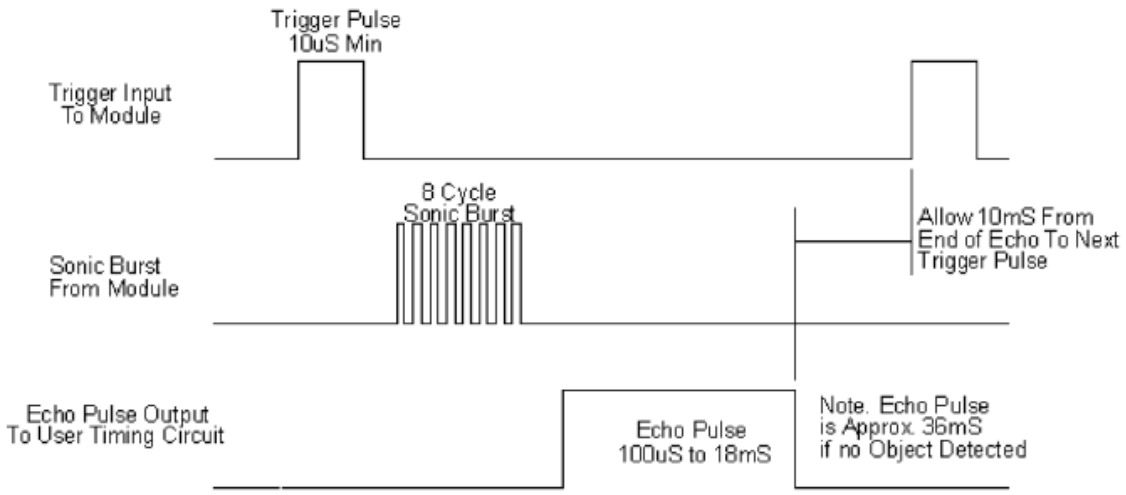
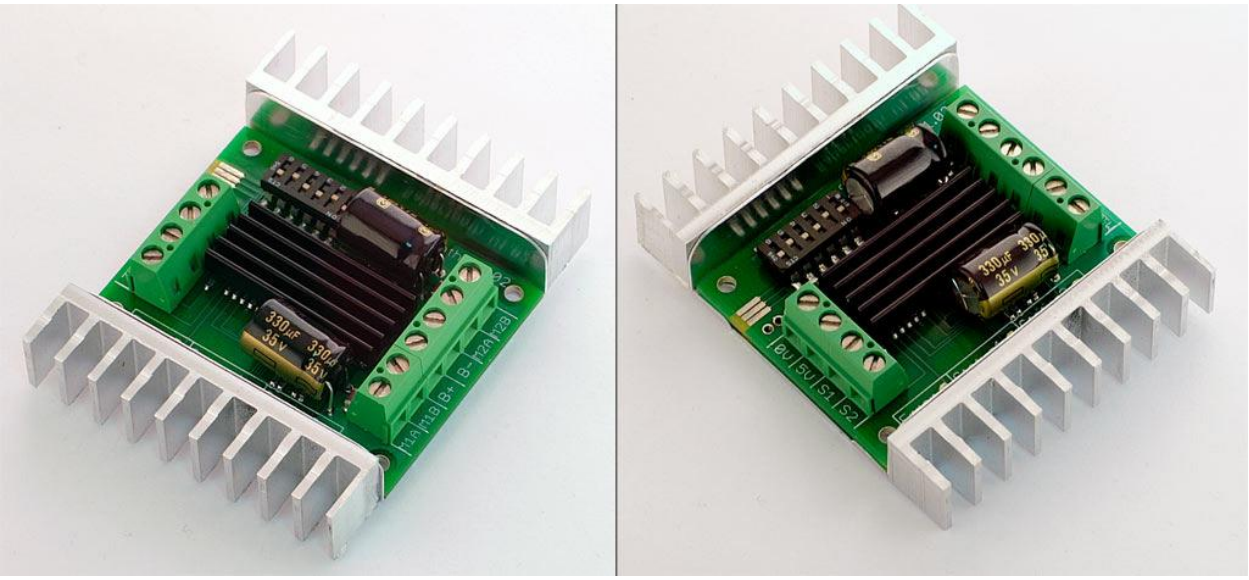


Figure 4. Operation of SRF-04 Ultrasonic sensor

In Hallway Navi Bot, a single trigger is used for both front and left ultrasonic sensors.

Sabertooth 2x10 motor controller:
It has 2 channel motor drive outputs and 5V digital command inputs. Table 5 shows commands for forward and reverse for left and right side motors. If ‘0’ command is sent, both the motors are stopped. These commands are transferred from MSP430 to Sabertooth motor controller through UART communication. Voltage translator is used to up the voltage to 5V for Sabertooth.



	Left motor	Right Motor
Full Forward	1	128
Zero Speed	64	192
Full Reverse	127	255

Applications:

Ultrasonic sensors have advantages for navigation tasks as they are unaffected by the lighting in the room. Navigation and collision avoidance programs is very useful in modern day robotics. It helps in preventing damage to hardware used in the machine. Collision avoidance programs has its' applications in modern commercial devices such as a smart room cleaner and a autonomous lawn mower.

Navigation Algorithm is useful for tracking new pathways in unknown areas and areas inaccessible to humans.

References:

[1] EE4735 Embedded System Engineering lecture slides by Prof. Kit Cischke.

[2] eZ430-RF2500 and MSP430x2xx Family User's Guide

[3] SRF-04 Ultrasonic sensor datasheet- http://inside.mines.edu/~whoff/courses/EENG383/lab/SRF04%20Technical%20Documentation.pdf

[4] Voltage Level Translator Datasheet http://www.ti.com/lit/ds/symlink/txs0104e.pdf

[5] Sabertooth 2x10 Motor Controller, Voltage Level Translator https://www.dimensionengineering.com/datasheets/Sabertooth2x10.pdf

Appendix:

//Code for Hallway Navi Bot

```
//**********************Initialization**********************//
#include "msp430x22x4.h"
#include "stdint.h"
volatile uint16_t left_sensor;
volatile uint16_t front_sensor;
volatile uint8_t i, j;
volatile uint8_t zone;
volatile uint8_t prev_z;
volatile uint8_t adj;
volatile uint8_t stop_count;
volatile prop = 1;
volatile uint8_t delay;
static uint8_t left_motor = 100;
static uint8_t right_motor = 228;
static uint8_t cal = 0;

#pragma vector=TIMERA1_VECTOR
__interrupt void IsrCntPulseTACC1 (void)
//-------------------------------------------------------------------------
// Func:  At TACCR1 IRQ, increm pulse count & toggle built-in Red LED
// Args:  None
// Retn:  None
//-------------------------------------------------------------------------
{
  switch (__even_in_range(TAIV, 10))  // I.D. source of TA IRQ
  {
   static uint16_t currEdgeStamp=10;
   static uint16_t prevEdgeStamp = 0;
   static uint32_t pulse_width_count=10;


    case TAIV_TACCR1:               // handle chnl 1 IRQ
      cal++;


         if ((cal/4)==0)
         {
         //Capturing the front sensor value

   currEdgeStamp = TACCR1;

   if ((P2IN&0x08)==0)
```

```
    {
     if (currEdgeStamp <= prevEdgeStamp)         // assuming period is smaller than FFFF
    {
       pulse_width_count = 0xFFFF - prevEdgeStamp + currEdgeStamp;
    }
    else
       pulse_width_count =  - prevEdgeStamp + currEdgeStamp;

    }

       if(cal/3)
    front_sensor = pulse_width_count/58;    // in cm
    prevEdgeStamp = currEdgeStamp;


          }



          //Navigation logic

if(cal/4)
{

//Taking Input from both front and left sensors and defining zones and conditions

if(front_sensor <= 45 || front_sensor > 660 && left_sensor > 25)
          zone = 0;
else if((front_sensor < 45 || front_sensor > 660) && left_sensor <= 25)
          zone = 6;
else
{
if(left_sensor > 25)
          zone = 1;
if(left_sensor > 100)
          zone = 2;
if(left_sensor > 125)
          zone = 3;
if(left_sensor > 150)
          zone = 4;
if(left_sensor > 200)
          zone = 5;
}
```

```
//Navigation Logic

if(stop_count >= 28)                        //STOP bot if trying to navigate near end of the
hallway
{
        while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
    UCA0TXBUF = 000;
        while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
      UCA0TXBUF = 000;
}
else
{
if(zone == 0)                                   //too close to obstacle, REVERSE & correct
{

        while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
    UCA0TXBUF = 000;                            // Stop Motors
        while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
      UCA0TXBUF = 000;                          // Stop Motors

        while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
      UCA0TXBUF = 32;                                   // LM = 1/2 reverse
        while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
      UCA0TXBUF = 160;                          // RM = 1/2 reverse

        __delay_cycles(4000);

        //while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
    //UCA0TXBUF = 000;
        //while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
      //UCA0TXBUF = 000;

        if(left_sensor <= 50)                       // if closer to left wall, then
REVERSE with a RIGHT TURN
        {
            while(!(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
            UCA0TXBUF = 48;                                 // LM = 1/4 reverse
            while(!(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
            UCA0TXBUF = 144;                        // RM = 3/4 reverse
            __delay_cycles(550);
            stop_count++;                           // INCREMENT Stop count
        }
        else                                                // if closer to right
wall, then REVERSE with a LEFT TURN
```

```
                {
                        while(!(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
                        UCA0TXBUF = 16;                                  // LM = 3/4 reverse
                        while(!(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
                        UCA0TXBUF = 176;                         // RM = 1/4 reverse
                        __delay_cycles(550);
                        stop_count++;                                    // INCREMENT Stop count
                }



}
if(zone == 5)                                           //new hallway detected, EXTREME LEFT
{
        left_motor = 100;
        right_motor = 228;

        //turn left



        for(i=0;i<10;i++)                                       // Spin ANTI-CLOCKWISE
        {
                left_motor = 1;                 //LEFT MOTOR = full reverse
            while ( !(IFG2 & UCA0TXIFG)) {};    // Confirm that Tx Buff is empty
       UCA0TXBUF = left_motor;

                 right_motor = 255;             //Right MOTOR = full forward
            while ( !(IFG2 & UCA0TXIFG)) {};    // Confirm that Tx Buff is empty
       UCA0TXBUF = right_motor;


        }
        __delay_cycles(400);
        stop_count = 0;                                         //reset Stop count

}
if(zone == 6)                                           //wall detected, REVERSE
{

                while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
      UCA0TXBUF = 000;
                while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
        UCA0TXBUF = 000;
```

```
                while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
        UCA0TXBUF = 32;                                 //1/2 reverse

                while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
        UCA0TXBUF = 160;                                //1/2 reverse

                __delay_cycles(5000);
                stop_count++;                                   // INCREMENT Stop count
}
if(front_sensor <= 150)                                 //obstacle detected in Zones 2 or 3
{
        if(zone == 2)                                   //if obstacle is detected in Zone 2
        {

              adj =0;
                        delay = 1;

                        if(front_sensor <= 120)
                                delay = 2;
                        else if(front_sensor < 90)
                                delay = 3;
                        else if(front_sensor < 60)
                                delay = 4;
                        //else if(front_sensor < 40)
                        //      delay = 5;
                        //else if(front_sensor < 40)
                        //      delay = 5;


                        // proportional drive = adj + min change + ((distance from obstacle)*(max
change - min change)/(difference in boundary limits))

                        prop = adj + 7 + ((150 - front_sensor)/21);                     //
proportional drive = adj + min change + ((distance from obstacle)*(12-7)/(150-45);


                for(j = 0;j <= delay; j++)
                        {
                                left_motor = 100;
                        right_motor = 228;

                                for(i=0;i <= prop; i++)                                 //TURN
LEFT
```

```
                                {
                                if(left_motor >= 75 && left_motor < 127)
                                {
                                left_motor = left_motor--;                  //INCREMENT LEFT
MOTOR
                                        while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff
is empty
                                UCA0TXBUF = left_motor;

                                }
                        if(right_motor > 203 && right_motor <= 255)
                                {
                                right_motor = right_motor++;                //DECREMENT right
MOTOR
                                while ( !(IFG2 & UCA0TXIFG)){};
                                        UCA0TXBUF = right_motor;
                                }
                        }
                        }

        }
        if(zone == 3)                                   // if obstacle detected in Zone 3
        {

                adj =0;
                delay = 1;

                if(front_sensor <= 120)
                        delay = 2;
                else if(front_sensor < 90)
                        delay = 3;
                else if(front_sensor < 60)
                        delay = 4;
                //else if(front_sensor < 40)
                //      delay = 5;
                //else if(front_sensor < 40)
                //      delay = 5;

                // proportional drive = adj + min change + ((distance from obstacle)*(max change
- min change)/(difference in boundary limits))

                prop = adj + 7 + ((150 - front_sensor)/21);                     // proportional drive
= adj + min change + ((distance from obstacle)*(12-7)/(150-45);
```

```
        for(j = 0;j <= delay; j++)
                {
                        left_motor = 100;
                right_motor = 228;

                        for(i=0;i <= prop; i++)                                 //TURN LEFT
                        {
                        if(left_motor >= 75 && left_motor < 127)
                        {
                        left_motor = left_motor--;                  //INCREMENT LEFT MOTOR
                                while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
                        UCA0TXBUF = left_motor;
                        }
                if(right_motor > 203 && right_motor <= 255)
                        {
                        right_motor = right_motor++;                  //DECREMENT right MOTOR
                        while ( !(IFG2 & UCA0TXIFG)){};
                                        UCA0TXBUF = right_motor;
                                }
                }
                }

        }
}
else                                                            //if no obstacle detected in Zones 2
or 3
{
if(zone == 2)                                           //Minor RIGHT
{
        adj =0;
        delay = 1;

        if(prev_z == 3)
                adj = 2;

        // proportional drive = adj + min change + ((distance from obstacle)*(max change - min
change)/(difference in boundary limits))

        prop = adj + 2 + ((125 - left_sensor)/20);                      // proportional drive = adj +
min change + ((distance from zone boundary)*(8-3)/(125-25));


   for(j = 0;j <= delay; j++)
```

```
        {
                left_motor = 100;
           right_motor = 228;

        for(i=0;i <= prop; i++)                                  //TURN RIGHT
        {
            if(left_motor >= 75 && left_motor < 127)
    {
      left_motor = left_motor++;                  //INCREMENT LEFT MOTOR
            while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
      UCA0TXBUF = left_motor;

    }
      if(right_motor > 203 && right_motor <= 255)
    {
      right_motor = right_motor--;                  //DECREMENT right MOTOR
      while ( !(IFG2 & UCA0TXIFG)){};
                UCA0TXBUF = right_motor;
         }

   }
        }


        if(prev_z ==0)
                stop_count++;                                    // INCREMENT Stop count
if previous zone is Zone 0

        prev_z = 2;

}
if(zone == 3)                                           //Minor LEFT
{
        adj =0;
        delay = 1;

//if(front_sensor <=150)                                  //obstacle in front or right of bot,
TURN LEFT

        //__delay_cycles(100);


        if(prev_z == 2)
                adj = 2;
```

```
        for(j = 0; j <= delay; j++)
        {
                left_motor = 100;
           right_motor = 228;

                // proportional drive = adj + min change + ((distance from obstacle)*(max change
- min change)/(difference in boundary limits))

                prop = adj + 3 + ((left_sensor - 125)/20);                      // proportional drive
= adj + min change + ((distance from zone boundary)*(8-3)/(125-25));

                for(i=0;i <= prop; i++)                                         //TURN LEFT
                {
                if(left_motor >= 75 && left_motor < 127)
                {
                left_motor = left_motor--;                  //INCREMENT LEFT MOTOR
                        while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
                UCA0TXBUF = left_motor;
                }
        if(right_motor > 203 && right_motor <= 255)
                {
                right_motor = right_motor++;                //DECREMENT right MOTOR
                while ( !(IFG2 & UCA0TXIFG)){};
                                UCA0TXBUF = right_motor;
                        }
        }
        }
        prev_z = 3;
        stop_count = 0;
}
}
if(zone == 1)                   //TURN HARD RIGHT
{
        adj = 0;

        left_motor = 100;
        right_motor = 228;

        if(prev_z = 2)
                adj = 5;
```

```
        // proportional drive = adj + min change + ((distance from obstacle)*(max change - min
change)/(difference in boundary limits))

        prop = adj + 3 + ((100 - left_sensor)*10)/75;                    // proportional drive = adj +
min change + ((distance from zone boundary)*(13-3)/(100-25));


        for(i=0;i <= prop; i++)                                          //TURN RIGHT
        {
            if(left_motor >= 75 && left_motor < 127)
    {
      left_motor = left_motor++;                  //INCREMENT LEFT MOTOR
            while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
      UCA0TXBUF = left_motor;

    }
      if(right_motor > 203 && right_motor <= 255)
    {
      right_motor = right_motor--;                  //DECREMENT right MOTOR
      while ( !(IFG2 & UCA0TXIFG)){};
                UCA0TXBUF = right_motor;
          }
           if (front_sensor < 50)
                __delay_cycles(10);
   }

        if(prev_z == 0)
        stop_count++;                                    // INCREMENT Stop count if previous
zone is Zone 0

        prev_z = 1;
}
if(zone == 4)                                            //TURN HARD LEFT
{
        adj = 0;

        left_motor = 100;
        right_motor = 228;

        if(prev_z == 3)
                adj = 5;

        // proportional drive = adj + min change + ((distance from obstacle)*(max change - min
change)/(difference in boundary limits))
```

```
        prop = adj + 3 + ((left_sensor - 125)*10)/75;                    // proportional drive = adj +
min change + ((distance from zone boundary)*(13-3)/(200-125));

        for(i=0;i <= prop; i++)                                          //TURN LEFT
        {
            if(left_motor >= 75 && left_motor < 127)
    {
      left_motor = left_motor--;                    //INCREMENT LEFT MOTOR
            while ( !(IFG2 & UCA0TXIFG)) {};   // Confirm that Tx Buff is empty
      UCA0TXBUF = left_motor;

    }
      if(right_motor > 203 && right_motor <= 255)
    {
      right_motor = right_motor++;                  //DECREMENT right MOTOR
      while ( !(IFG2 & UCA0TXIFG)){};
                UCA0TXBUF = right_motor;
         }

   }

        prev_z = 4;
        stop_count = 0;                                                  //reset Stop count
}

}
prev_z = zone;                                                           // store previous zone value
cal=0;                                                                   //reset cal
}


   TACCTL1 &= ~0x01;
   break;

   case TAIV_TACCR2:              // chnl 2 IRQ
   case TAIV_TAIFG:               // ignore TAR rollover IRQ
   default:                    // ignore everything else
  }

  //TACCTL1 &= ~0x01;              //clear interrupt flag
  }

#pragma vector=TIMERA0_VECTOR
```

```
__interrupt void Timer_A (void)
{
   static uint16_t currEdgeStamp_2=10;
   static uint16_t prevEdgeStamp_2 = 0;
   static uint32_t pulse_width_count_2=10;

    //case TAIV_TACCR1:                // handle chnl 1 IRQ

   if ((cal/3)==0)
          {
   currEdgeStamp_2 = TACCR0;

    if ((P2IN&0x04)==0)
    {
     if (currEdgeStamp_2 <= prevEdgeStamp_2)         // assuming period is smaller than FFFF
    {
        pulse_width_count_2 = 0xFFFF - prevEdgeStamp_2 + currEdgeStamp_2;
    }
    else
        pulse_width_count_2 =  - prevEdgeStamp_2 + currEdgeStamp_2;

    }

    if(cal/2)
    left_sensor= pulse_width_count_2/58;  // in cm
    prevEdgeStamp_2 = currEdgeStamp_2;
       }
    TACCTL0 &= ~0x01;

}


void InitPorts (void)
//------------------------------------------------------------------------
// Func:  Initialize the ports for I/O on TA1 Capture
// Args:  None
// Retn:  None
//------------------------------------------------------------------------
{
   P2DIR &= ~0x08;                    // P2.3 = Input mode
   P2SEL |= 0x08;                    // Timer A2 select
   P2DIR &= ~0x04;                    // P2.4 = Input mode
   P2SEL |= 0x04;                    // Timer A2 select
   P4DIR |= 0x10;                   // P4.4 = Output mode
```

```
   P4SEL |= 0x10;                    // P4.4 = TB1 = TB compare OUT1
   P3SEL = 0x30;                      // P3.4,5 = USCI_A0 TXD/RXD
}



void main( void )
//-------------------------------------------------------------------
// Func:  Configure UART  functions & Ports,
//        Configure Timer functions & params,
//        Go to sleep & let Timer ISR do all the work
// Args:  None
// Retn:  None
//-------------------------------------------------------------------
{ WDTCTL = WDTPW + WDTHOLD;          // Stop WDT

   InitPorts();

  BCSCTL1 = CALBC1_1MHZ;              // DCO = 1 MHz
  DCOCTL  = CALDCO_1MHZ;              // DCO = 1 MHz
  UCA0CTL1 |= UCSSEL_2;               // UART use SMCLK

  UCA0MCTL = UCBRS0;                  // Map 1MHz -> 9600 (Tbl 15-4)
  UCA0BR0  = 104;                     // Map 1MHz -> 9600 (Tbl 15-4)
  UCA0BR1  = 0;                       // Map 1MHz -> 9600 (Tbl 15-4)

  UCA0CTL1 &= ~UCSWRST;               // Enable USCI state mach

  while ( !(IFG2 & UCA0TXIFG)) {};    // Confirm that Tx Buff is empty
  UCA0TXBUF = left_motor;             // Init robot to stopped state
  while ( !(IFG2 & UCA0TXIFG)) {};    // Confirm that Tx Buff is empty
  UCA0TXBUF = right_motor;            // Init robot to stopped state

  TBCCR0 = 40250-1;              // 35 ms Set frequency of PWM (UP mode)
  TBCCR1 = 10;          // PW = 10us Set pulse wid of PWM (UP mode) initialize to 0 degree

  TBCTL   = TBSSEL_2 | ID_0 | MC_1 ;   // Sel. SMCLK | div by 1 | UP Mode | enable TAIE
  TBCCTL1 = CCIS0 | OUTMOD_6;         // Chnl-1 Inp=CCI1B | OUTMOD=Tgl/Rst
                              // Note:  Compare mode is default
  TACTL   = TASSEL_2 | ID_0 | MC_2;          // SMCLK | Div by 1 | Contin Mode
  TACCTL1 = CM0 |CM1| CCIS0 | CAP | SCS |SCCI| CCIE;  // Ris Edge | inp = CCI1B |
                                  // Capture | Sync Cap | Enab IRQ    CM1 = rising mode bit 01
  TACCTL0 = CM0 |CM1| CCIS0 | CAP | SCS |SCCI| CCIE;
```

```
  _BIS_SR(LPM1_bits + GIE);           // Enter LPM1 w/ IRQs enab

  // The CPU is now asleep in LPM3 with general IRQs enabled (GIE=1).
  // The CPU never wakes up because all the work is done by peripheral
  // hardware (Timer A Compare Channel 1 signal OUT1).
}
```