Project Report

**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 | 14 |
| 13 | Appendix (with code) | 15 |

**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:**

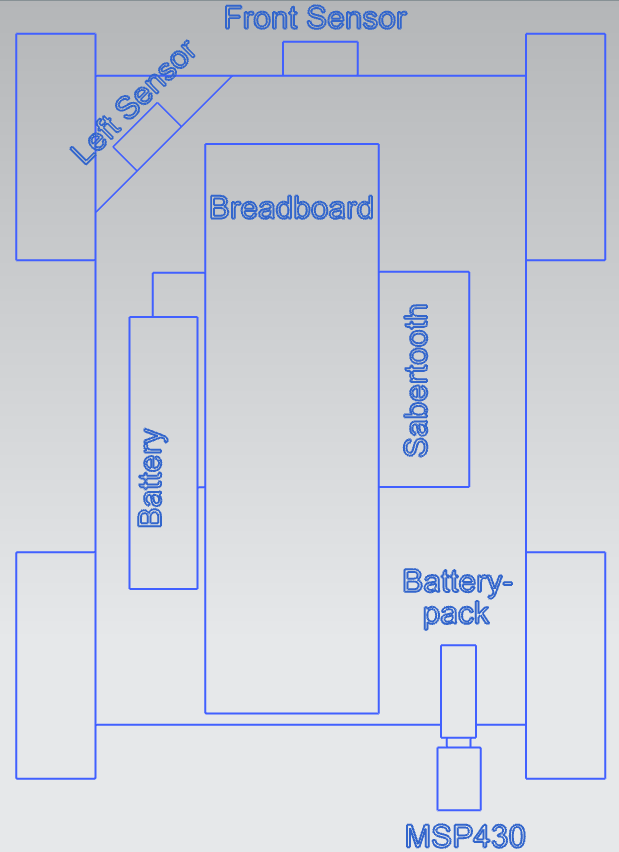


Figure 1.1 Robot layout with components

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

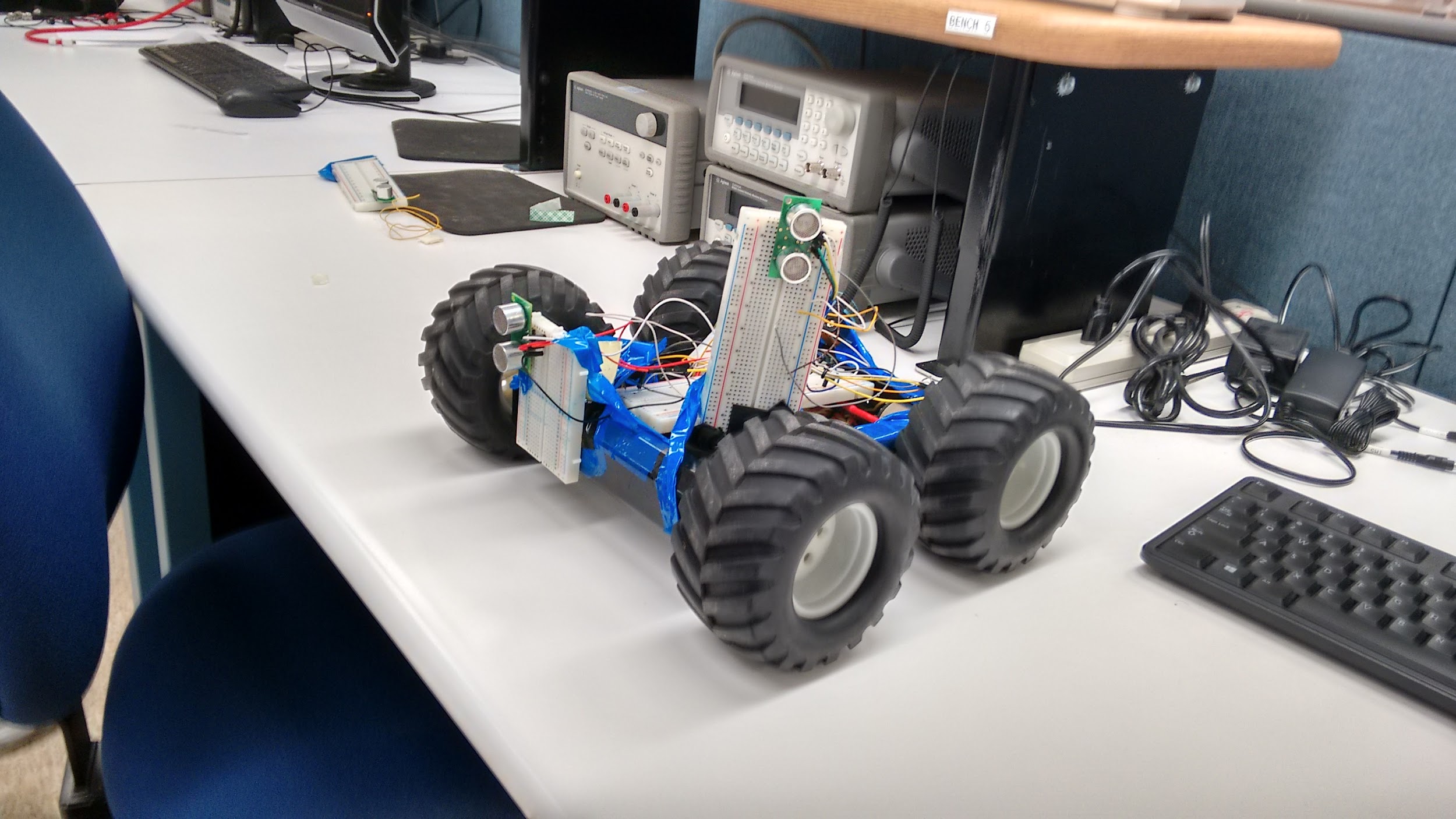


Figure 1.3 Hallway navi bot

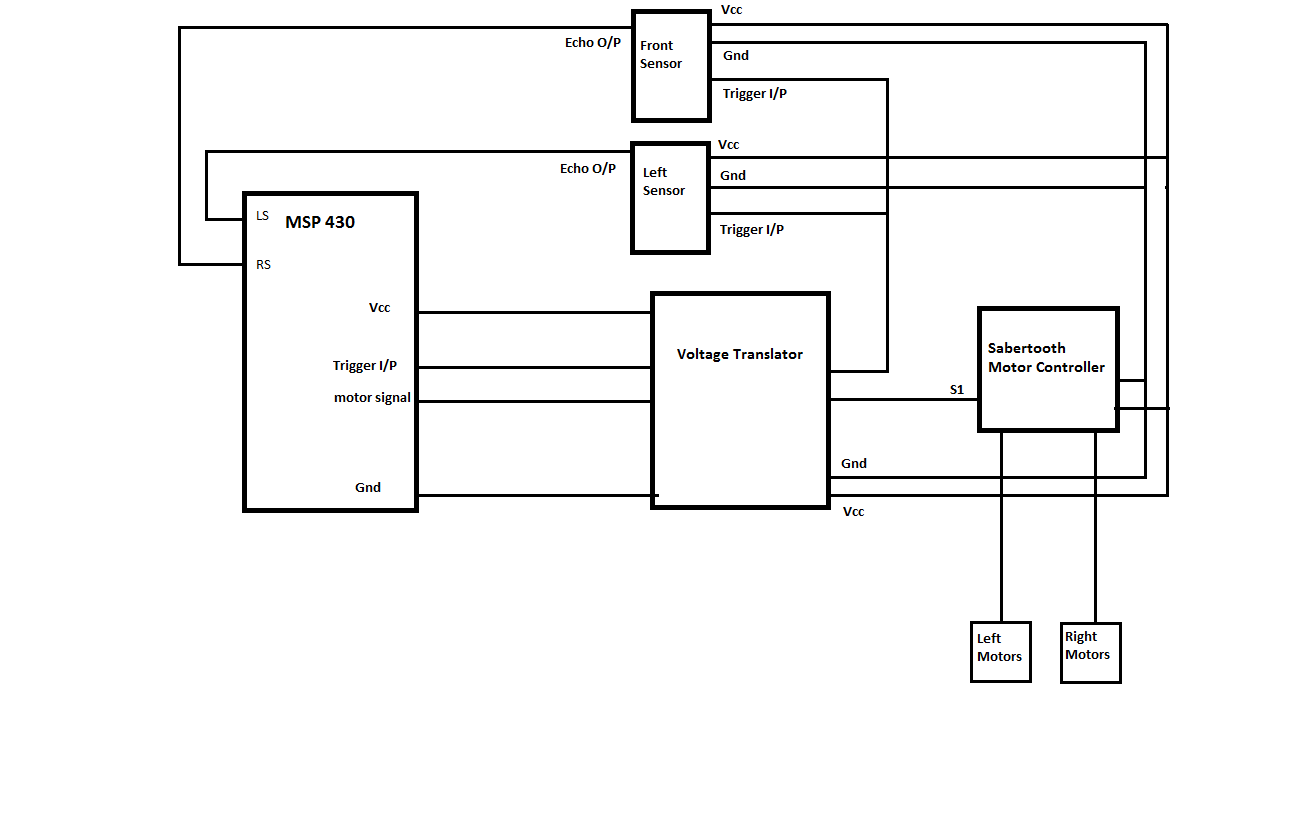
**Pin Diagram**

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

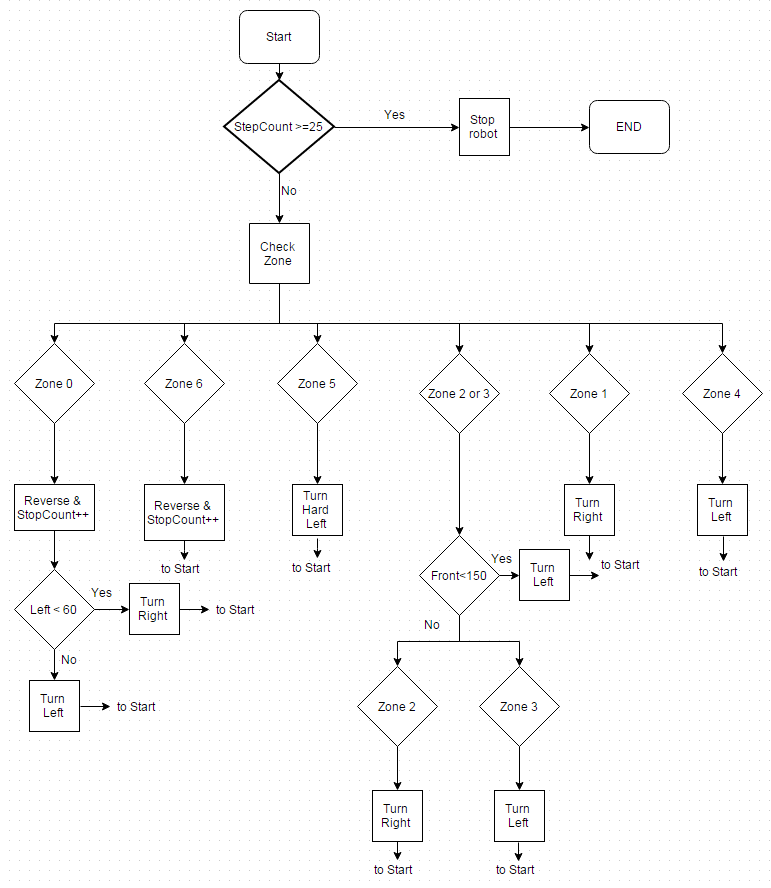
With zone 2 and 3 being the central zones of the hallway, the bot is to try and maintain itself between these zones.

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:**

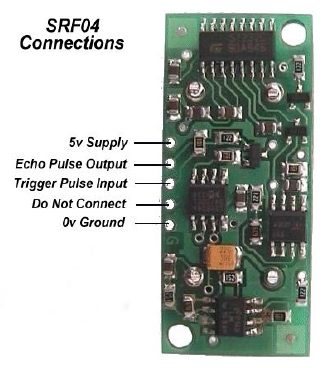


Figure 3. Pin connection for SRF-04 Ultrasonic sensor

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.

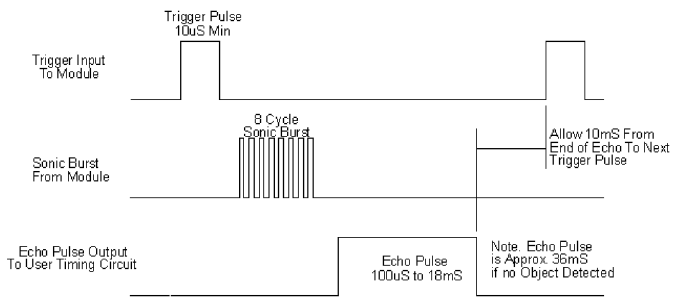
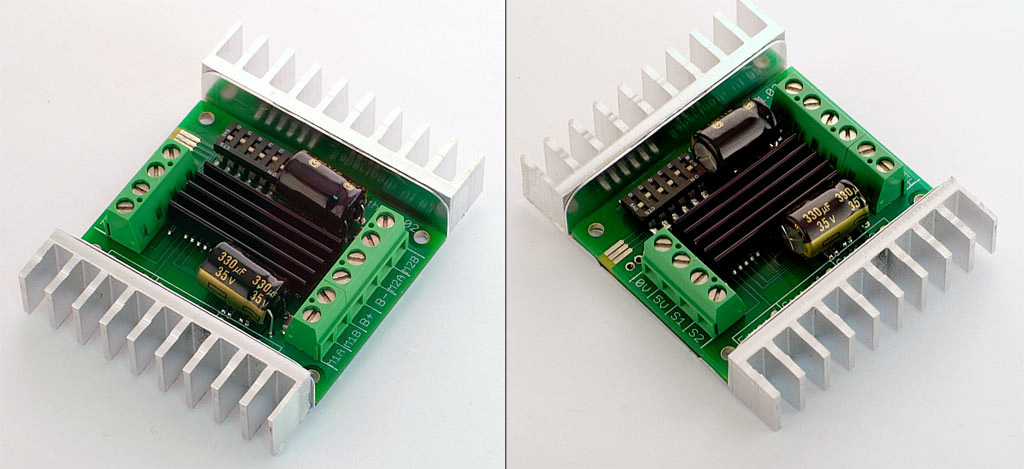


Figure 4. Operation of SRF-04 Ultrasonic sensor

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] eZ430­RF2500 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**

#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)

{

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;

}

if(stop\_count >= 28)

{

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;

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(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)

{

while(!(IFG2 & UCA0TXIFG)) {};

UCA0TXBUF = 48;

while(!(IFG2 & UCA0TXIFG)) {};

UCA0TXBUF = 144;

\_\_delay\_cycles(550);

stop\_count++;

}

else

{

while(!(IFG2 & UCA0TXIFG)) {};

UCA0TXBUF = 16;

while(!(IFG2 & UCA0TXIFG)) {};

UCA0TXBUF = 176;

\_\_delay\_cycles(550);

stop\_count++;

}

}

if(zone == 5) //new hallway detected, EXTREME LEFT

{

left\_motor = 100;

right\_motor = 228;

//turn left

for(i=0;i<10;i++)

{

left\_motor = 1; //INCREMENT LEFT MOTOR

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

UCA0TXBUF = left\_motor;

right\_motor = 255; //INCREMENT Right MOTOR

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

UCA0TXBUF = right\_motor;

}

\_\_delay\_cycles(400);

stop\_count = 0;

}

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++;

}

if(front\_sensor <= 150) //obstacle detected

{

if(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;

prop = adj + 7 + ((150 - front\_sensor)/21); //(10-5)/(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;

}

}

}

//\_\_delay\_cycles(100);

}

}

if(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;

prop = adj + 7 + ((150 - front\_sensor)/21); //(10-5)/(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(zone == 2) //Minor RIGHT

{

adj =0;

delay = 1;

if(prev\_z == 3)

adj = 2;

prop = adj + 2 + ((125 - left\_sensor)/20); //(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++;

}

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;

prop = adj + 3 + ((left\_sensor - 125)/20); //(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;

prop = adj + 3 + ((100 - left\_sensor)\*10)/75; //(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;

}

\_\_delay\_cycles(5);

if (front\_sensor < 50)

\_\_delay\_cycles(5);

}

if(prev\_z == 0)

stop\_count++;

prev\_z = 1;

}

if(zone == 4) //TURN HARD LEFT

{

adj = 0;

left\_motor = 100;

right\_motor = 228;

if(prev\_z == 3)

adj = 5;

prop = adj + 3 + ((left\_sensor - 125)\*10)/75; //(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;

}

}

prev\_z = zone;

cal=0;

}

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: Initialize & start timer A to generate a PWM waveform &

// output it to P2.3/TA1 without further CPU intervention.

// 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).

}