Sensor-Controlled Robot

Rodrigue de Schaetzen

December 15, 2019

Abstract

A simple model of an automated emergency braking system was made using the ultrasonic module HC-SR04, a TI (Texas Intrument) MSP430 16-bit micro-controller, and an Ivolador motor robot car chassis kit. Car braking was simulated by cutting off power to the DC motors when distance measurement reached a certain value. The minimal lag distance (distance traveled by car after reaching distance threshold) was found to be 2 cm or 10% of the car's length. A back-up beeper functionality was incorporated and a simple algorithm allowed the model car to maneuver around obstacles. Sensor range was found to be $3cm - 250cm$. Lastly, serial communication was used to record data from the sensor throughout the trajectory of the car.

1 Introduction/Background

An autonmous or self driving car is defined as a vehicle that is capable of sensing its environment and navigating without human input [1]. In recent years the automotive industry has seen immense growth in the number of commercially available self-driving cars. This race to achieve full autonomy is split between numerous different companies such as Google, Uber, and Tesla.

At the basic level a self driving system works by collecting data on the surroundings using several sensors, and localization systems in order to produce an appropriate response. One of the benefits of having a self-driving system is the increase in response time (compared to human reaction time) when faced with an unexpected obstacle on the road. This was the motivation for the main purpose of this project; to model a basic automatic braking system and some sort of guidance system. Specifically, an ultrasonic distance measurement sensor at the front of the car was used to relay data on the surroundings to the micro-controller.

2 Apparatus

The first part of the construction was putting together all the components of the car kit. A breadboard was then attached to the front of the chassis along with the ultrasonic sensor and a piezoelectric buzzer. An iPhone external battery (5 V) was used to power the MSP430 and to give the breadboard a ground and +5 V source. The included 6 V battery pack powers the drivers (V_{CC2} - pin 8) in the h-bridge which powers the motors.

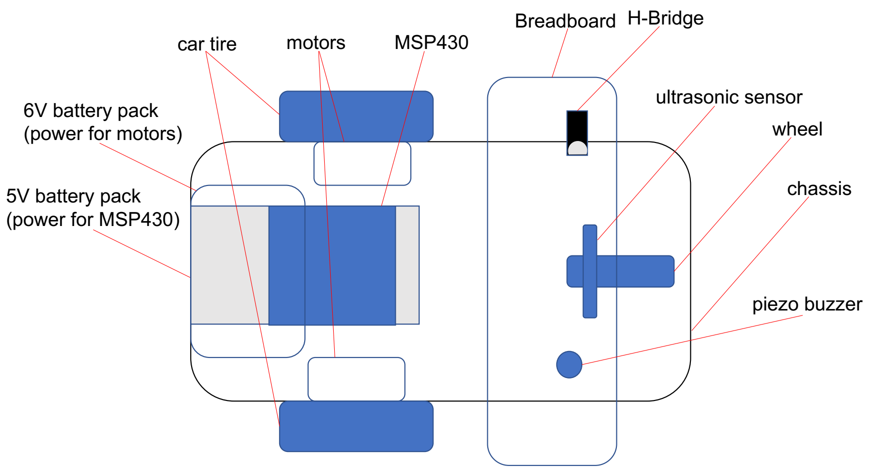
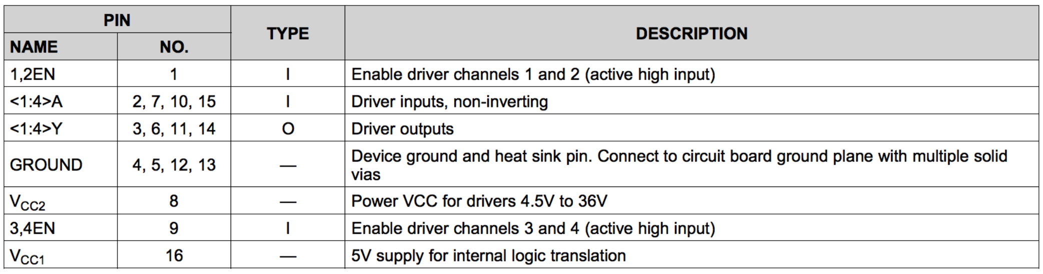


Figure 1: Diagram of the model car with all the active components.

Appropriate wire connections were then made between all electrical components. The use of an h-bridge allowed for bidirectional current, hence forward and reverse drive modes were possible.



PIN		TYPE	DESCRIPTION
NAME	NO.		
1,2EN	1	I	Enable driver channels 1 and 2 (active high input)
<1:4>A	2, 7, 10, 15	I	Driver inputs, non-inverting
<1:4>Y	3, 6, 11, 14	O	Driver outputs
GROUND	4, 5, 12, 13	—	Device ground and heat sink pin. Connect to circuit board ground plane with multiple solid vias
V_{CC2}	8	—	Power VCC for drivers 4.5V to 36V
3,4EN	9	I	Enable driver channels 3 and 4 (active high input)
V_{CC1}	16	—	5V supply for internal logic translation

Figure 2: Pin functions of H-Bridge (SN754410)

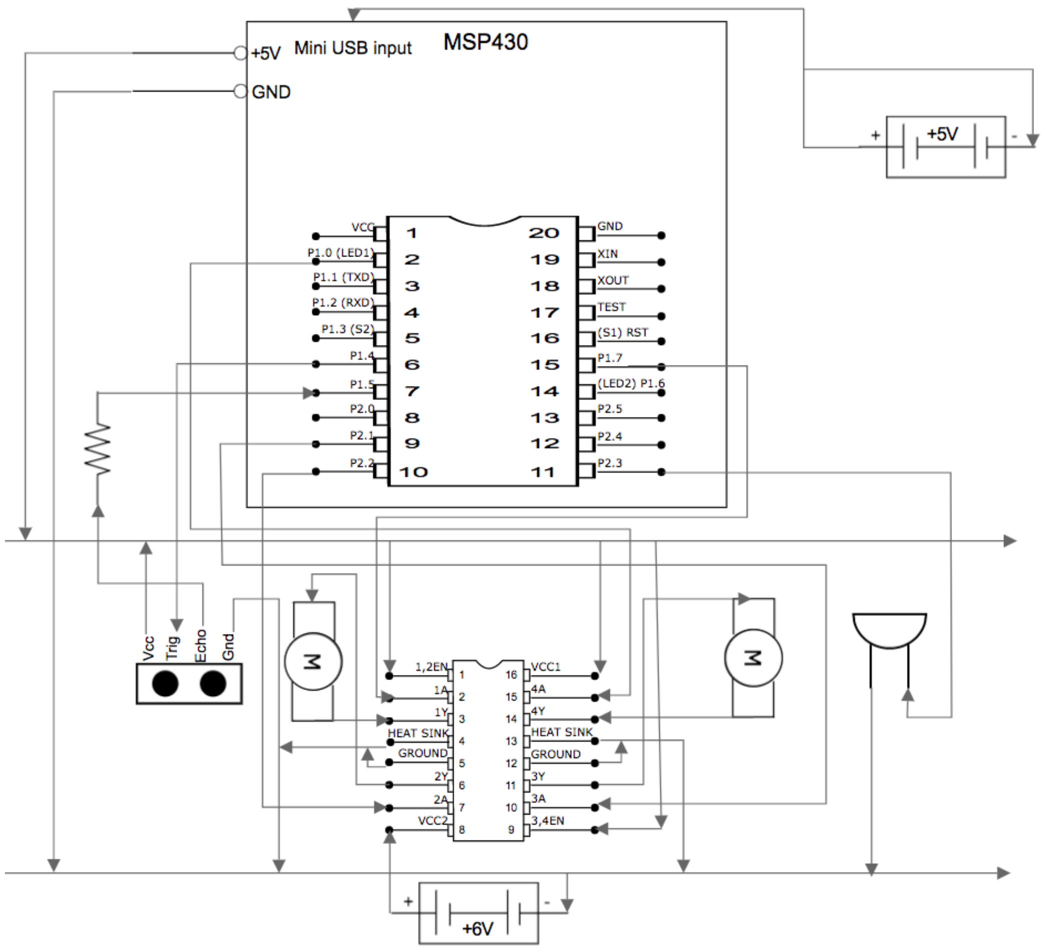
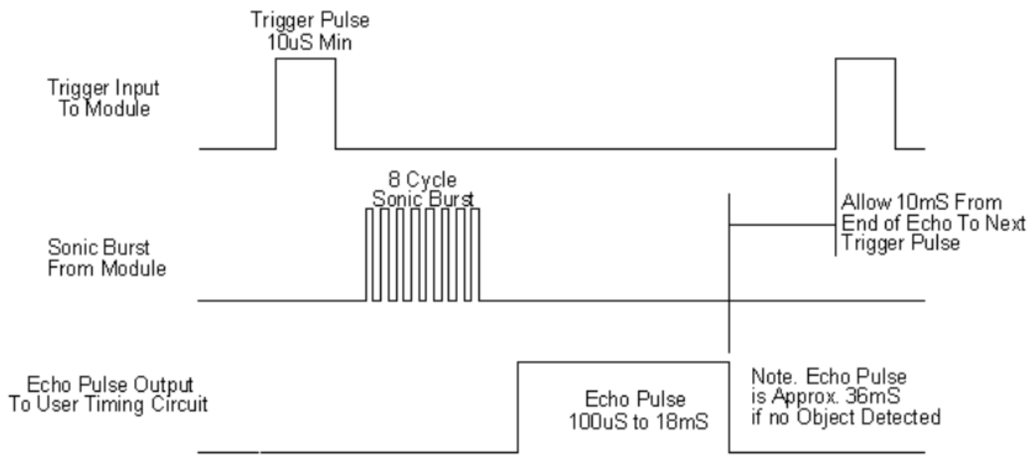


Figure 3: Schematic of all electrical components.

3 Procedure

3.1 Sensor

The SRF04 is an ultrasonic sensor which uses sonar to determine the distance to an object. According to product specifications, it has a range of $2cm - 400cm$ with an accuracy of $3mm$ [2]. A measurement is initiated by the micro-controller which sends a $10\mu s$ pulse to the trigger pin. This signals the module to send a high-frequency wave which then triggers the echo pin. A configured timer on the MSP430 counts while the echo pin is at a high logic state. Once the reflected wave is received or the echo pulse has timed out at $30mS$ the echo pin returns to a low logic state and the timer stops. A conversion factor gives the distance measurement in centimeters.



Trigger Pulse
10uS Min
Trigger Input To Module
8 Cycle Sonic Burst
Sonic Burst From Module
Allow 10mS From End of Echo To Next Trigger Pulse
Echo Pulse Output To User Timing Circuit
Echo Pulse 100uS to 18mS
Note. Echo Pulse is Approx. 36mS if no Object Detected

Figure 4: Ultrasonic sensor (SRF04) timing diagram.

3.2 Serial communication

UART or Universal Asynchronous Receiver/Transmitter is the communication protocol used to receive data on the computer through the MSP430. A byte is transmitted by writing to UCA0TXBUF. Serial communication in this project was used to record data from the sensor as the car operates.

3.3 Motor operation

The two motors were controlled independently by the MSP430. As seen in the schematic, pins 2 and 7 on the h-bridge connect to pins P1.7 and P2.2 respectively on the micro-controller which controls the left motor. A similar setup was true for the right motor. To drive both motors forward pins 7 and 10 were set to high by writing a high logic state to P2OUT and the pins 2 and 15 were set to low by writing a low logic state to P1OUT. The reverse is true to drive to motors in reverse. Turning in either direction was achieved by powering only one motor.

3.4 Beeper

A piezoelectric buzzer was used to simulate a vehicle back-up beeper. It worked by pulse width modulation or PWM. The MSP430 sent a signal at regular intervals which produced a tone determined by the duty cycle and period of the pulse.

3.5 Car operation

To power the entire system, plug the USB cable into the MSP430 and press the side button on the iPhone battery pack. To record data from the sensor using serial communication, plug the USB cable into the computer and then launch the Python script. To initiate the programmed commands press button P1.3 on the MSP430. To stop operation at any time press the RESET button. Below are the steps of the program:

1. Drive forward.
2. If measurement is less than distance threshold, then go to step 3. Otherwise, send measurement to computer and go back to step 1.
3. Drive backward while beeping twice.
4. Stop. Rotate approximately 18 degrees and record distance measurement on the MSP430.
5. Repeat step 4 19 times for a total of 20 measurements (and for a full 360 degrees).
6. Drive forward in the direction with the greatest distance measurement.
7. Repeat steps 1-6.

4 Results

4.1 Performance

4.1.1 Phase 1

In the first phase the aim was to get the motors to work simultaneously with the ultrasonic sensor. This would test the accuracy of the sensor and the overall response time of the automated braking system. Note that the sensor is placed 3 cm from the front-end of the car chassis and the threshold distance refers to the distance at which power is cutoff to the motors.

From the plots, a decrease in delay time between each measurement gives a faster braking-response. This delay was reduced to a minimum of $10mS$ stated by the SRF04 timing diagram. The lowest tested threshold without the car hitting the obstacle was found to be 6 cm with a sensor-determined distance of 4 cm. Hence 2 cm is the minimal lag distance for this automated braking system meaning that when an object is detected and the measured distance has reached the threshold, the car will move at least 2 cm before

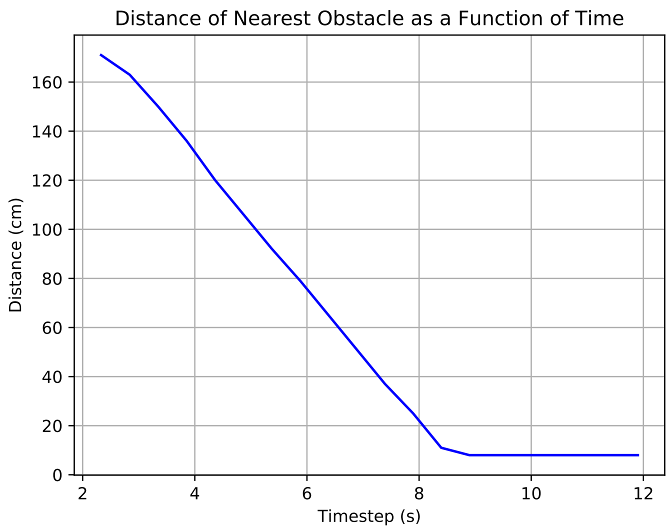


Figure 5: Transmitted sensor data with 20 cm distance threshold. The stopping distance differs by 10 cm which is partially due to the programmed delay of 0.5 s after each transmitted distance.

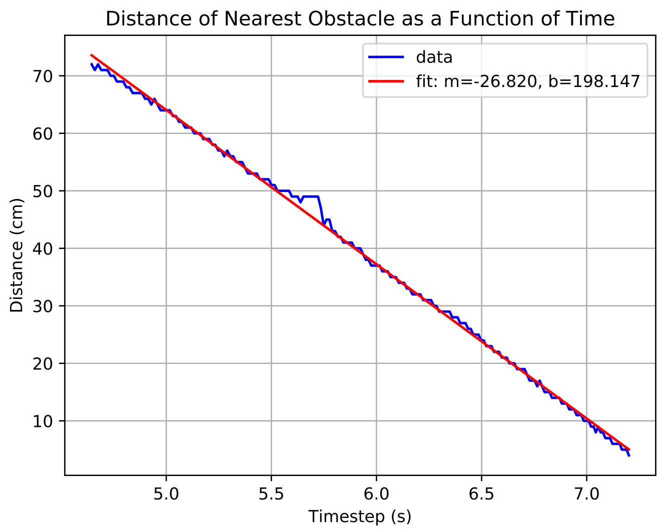


Figure 6: Fitted transmitted sensor data with 6 cm distance threshold. Delay after each transmitted distance is 0.01s. The stopping distance is 4 cm which is very close to the threshold. The speed of the car given by the fit is $0.2682 m/s$ or roughly $1 km/h$

stopping. In comparison to the dimensions of the model car, this is 10% of the length.

4.1.2 Phase 2

Next, a back-up beeper and a simple guidance system was implemented to add more self-driving like features to the model car. The back-up beeper simulated by the piezoelectric buzzer worked as expected, beeping exactly twice at a specific tone while the car went in reverse. For the guidance system, the goal was for the car to be able to determine the direction in which obstacles are the furthest away based on a 360 degree scan consisting of 20 distance measurements. The corridors and common space of my apartment (Walter Gage residence) was used as testing grounds for this algorithm. From the trajectories shown on figures 8 and 10 it is clear the car successfully goes in a direction where obstacles are far away. Moreover, the similarity in the end path (specifically steps 2, 3 on figure 8 and steps 3, 4 on figure 10) demonstrate the effectiveness of the scan. Note scattering of some of the data is most likely due to the unevenness of some of the obstacles such as wall corners, door frames etc.

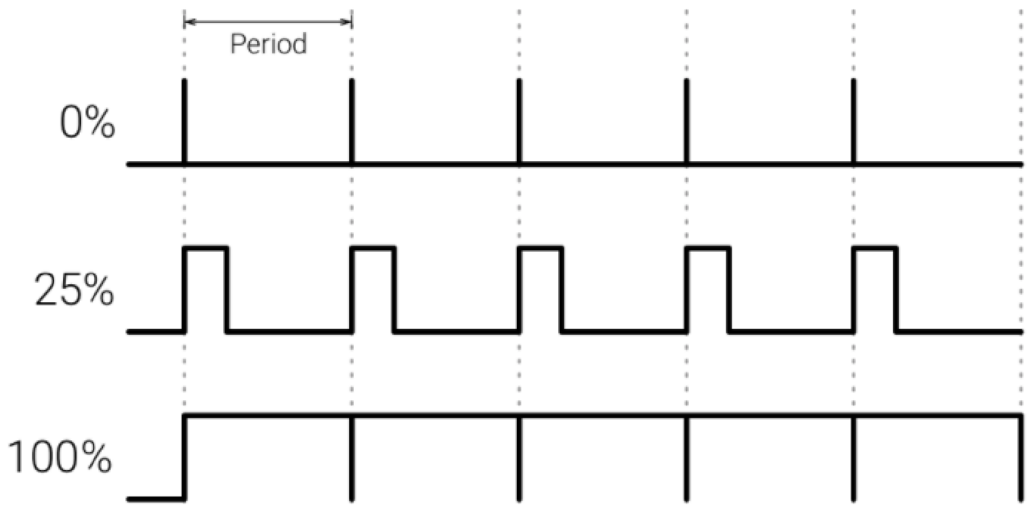


Figure 7: Pulse Width Modulation (PWM) with different duty cycles. The beeper has a 30% duty cycle and a period of 1000-1. It has a fixed on/off interval of 0.4 s on and 0.2 s off.

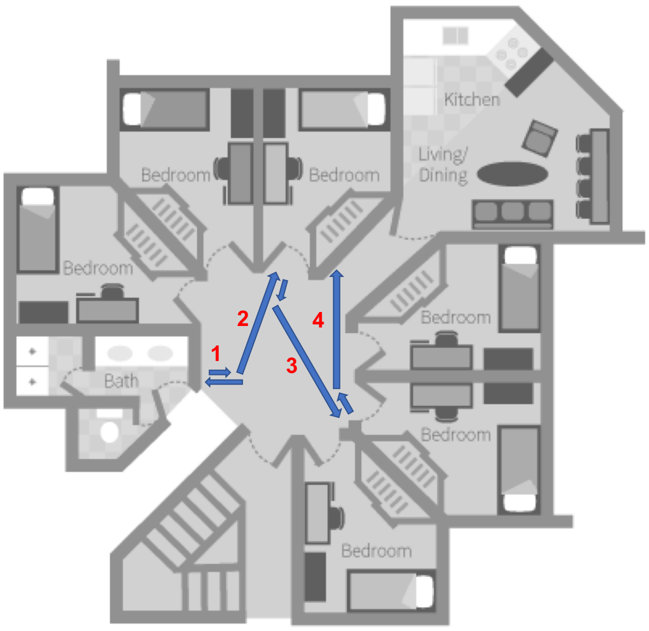


Figure 8: First set of data: trajectory of model car indicated by arrows with start of path at 1 and end of path at 4.

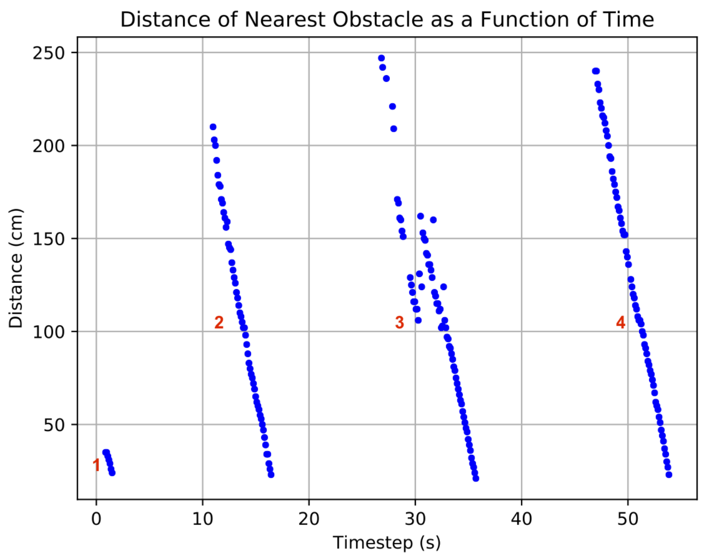


Figure 9: Distance data throughout car trajectory. Gap between each step is due to reverse + beep mode and scanning of area.

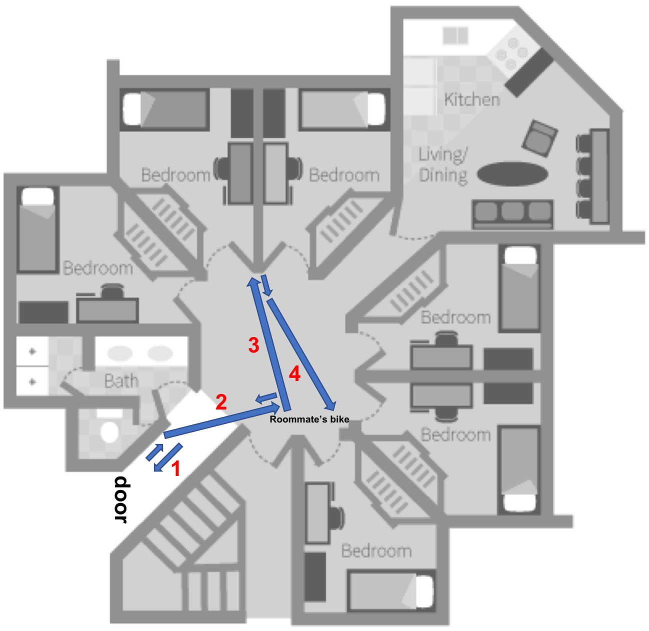


Figure 10: Second set of data: trajectory of model car indicated by arrows with start of path at 1 and end of path at 4.

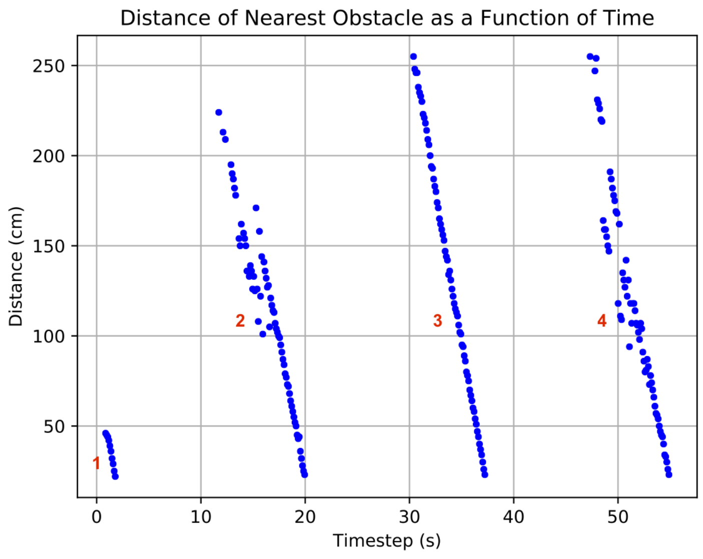


Figure 11: Distance data throughout car trajectory. Gap between each step is due to reverse + beep mode and scanning of area.

4.2 Encountered problems

4.2.1 Motor speed difference

In the initial setup, the car did not drive straight. Instead, it would slightly turn to the right indicating a difference in speed between the two wheels. Specifically the left wheel drove at a faster rate than the right wheel. Further investigation showed there was a difference of 0.2 V. A resistor of 6Ω was placed between pin 3 on the h-bridge and the left motor input which evened out the speed.

4.2.2 Range of sensor

As stated already, the supposed range of the sensor is $2cm - 400cm$, so for any distance greater than 4 m the distance measurement will be returned as 0 due to the echo timeout after $10mS$. Though from the data, the max distance found while navigating through the room was 250 cm, therefore the upper bound on the range is far less than the stated value. This restricting range limited the robot's ability to drive to more obscure places such as the dining room and kitchen.

5 Discussion

5.1 Limitations

No matter how small the programmed delay is between consecutive measurements, there will always be an intrinsic delay due to the speed of sound. Of course for robot applications and other slow moving devices this effect would be negligible. But for motor vehicles driving at highway speeds the delay would be significant thus rendering sonar as an ineffective system for detecting obstacles.

5.2 Extensions

Below is a list of possible extensions in the design and construction of this project:

- Another h-bridge to allow simultaneous forward and backward driving modes (e.g. right wheel driving forward and left wheel driving backward) for more precise maneuvering.
- Use PWM for smooth decelerating braking (viable if speed of model car is boosted)

- Addition of sensors on the back and sides of the robot.
- Use a wireless communication protocol such as Bluetooth, and IP-based communication.

6 Conclusions

This project successfully built an automated breaking system using data from an ultrasonic sensor. The minimum lag distance was found to be 2 cm or 10% of the robot's length. The addition of a back-up beeper gave the model car an added level of imitation to full scale vehicles by using PWM. Finally, the algorithm that guided the car's trajectory successfully maneuvered to places of the room where obstacles are further away based on a 360 degree scan of its surroundings.

References

[1] What is an Autonomous Car? - Definition from Techopedia
`https://www.techopedia.com/definition/30056/autonomous-car`

[2] Ultrasonic Sensor - HC-SR04
`https://www.sparkfun.com/products/13959`

7 Appendix

Following this page is the fully commented code used for this project.

```
//  motorsensor.c
#include "msp430.h"

// define all used ports on MSP430
#define     M1_1                  BIT1
#define     M1_2                  BIT2
#define     M2_1                  BIT7
#define     M2_2                  BIT1

#define     TRIG                  BIT4
#define     ECHO                  BIT5

#define     TXD                   BIT2

#define     BUTTON                BIT3

#define     BEEPER                BIT6

// define all variables
long sensor;
unsigned int distance;

unsigned int d;
unsigned int max;
unsigned int ind;

// define all methods
void beep(void);
void scan(int);
int measuredistance(void);

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;           // Stop WDT

    /* next three lines to use internal calibrated 1MHz clock: */
    BCSCTL1 = CALBC1_1MHZ;              // Set range
    DCOCTL = CALDCO_1MHZ;
    BCSCTL2 &= ~(DIVS_3);               // SMCLK = DCO = 1MHz

    // setup port for MOTORS, TRIG, BEEPER, ECHO, and BUTTON
    // set all ports to output except for P1.5 which is for ECHO and
    // P1.3 which is for push-button interrupt
    P1DIR = 0xDF;
    P1DIR &= ~BUTTON;
    P2DIR = 0xFF;

    // setup port for txd
    P1DIR |= TXD;
    P1OUT |= TXD;

    P1REN |= ECHO + BUTTON;             // Enable internal pull-up resistor on
    P1OUT &= ~(M1_1 + M2_1);
    P1OUT |= BUTTON;
    P2OUT &= ~(M1_2 + M2_2);
    P1IE |= BUTTON;                     //Enable input at P1.3 as an interrupt

    /* Configure hardware UART */
    P1SEL  |= TXD + BEEPER;     // P1.2=TXD and BEEPER to TA0.1
    P1SEL2 |= TXD;              // P1.2=TXD
    UCA0CTL1 |= UCSSEL_2;       // Use SMCLK
    UCA0BR0 = 104;              // Set baud rate to 9600 with 1MHz clock
    UCA0BR1 = 0;
    UCA0MCTL = UCBRS0;          // Modulation UCBRSx = 1
    UCA0CTL1 &= ~UCSWRST;       // Initialize USCI state machine

    _BIS_SR (GIE);              // Enable general interrupt
}
```

```
// Port 1 interrupt service routine
void __attribute__ ((interrupt(PORT1_VECTOR)))  PORT1_ISR(void)
/* Main Application Loop */ {
    while(1){

       int d = (int)measuredistance(); // make distance measurement with sensor

        if(d > 20){                    // drive forward if distance > 20cm
            P1OUT &= ~(M1_1 + M2_1);
            P2OUT |= M1_2 + M2_2;
        }

        else{
            P2OUT &= ~(M1_2 + M2_2);   // otherwise drive in reverse
            P1OUT |= M1_1 + M2_1;      // while buzzer beeps twice
            beep();
            P1OUT &= ~(M1_1 + M2_1);
            __delay_cycles(500000);    // stop car
            scan(d);                   // start scanning environment
        }

        while (!(IFG2 & UCA0TXIFG));   // wait for TXD to be ready to send data
        UCA0TXBUF = d;                 // transmit data back to host computer for python to print
        __delay_cycles(100000);        // give the process some time to work

        P1IFG = 0x00;                  // clear any interrupt flags

    }
}

// do a 360 scan of environment and find direction
// in which obstacles are the furthest away
void scan(int distance)
{
    int arr[20] = {distance};          // initialize an array of size 20 with first element
                                       // in array being original distance measurement
    for (int i = 1; i < 20; i++)       // iterate over each item in array starting at index=1
    {
        P2OUT |= M1_2;                 // turn right for 0.14s (value found through rigorous
testing)
        __delay_cycles(140000);        //***
        P2OUT &= ~(M1_2 + M2_2);       // stop
        int d = (int)measuredistance(); // take distance measurement and insert it into array
        arr[i] = d;
        __delay_cycles(200000);        // give a delay between each measurement
    }                                  // after for loop sensor would have taken 20 measurements
                                       // each at around 18 degree intervals for a full
                                       // 360 degree scan

    int max = 0;                       // initialize max distance = 0
    int ind = 0;                       // initialize array index = 0

    for(int i = 0; i < 20; i++)        // find element in array with greatest distance
    {
        if(max < arr[i]) {             // this will give the index of the array with the
            max = arr[i];              // greatest distance which will be used to find the
            ind = i;                   // direction in which the measurement was made
        }
    }

    if(ind < 10) {                      // if the index is less than 10 then turn right ind
        P2OUT |= M1_2;                  // number of times
        for(int i = 0; i < ind; i++)    // this will make the car face in the direction in
        {                               // which the max distance measurement was found
            __delay_cycles(140000);     //***delay here is the same when measurement was made***
        }
        P2OUT &= ~(M1_2 + M2_2);
    }

    else
    {                                  // if the index is greater or equal to 10 then turn left
        P2OUT |= M2_2;                 // (20-ind) number of times
        for(int i = 0; i < (20-ind); i++) // this will make the car face in the direction in
        {                              // which the max distance measurement was found
            __delay_cycles(140000);     //***delay here is the same when measurement was made***
        }
        P2OUT &= ~(M1_2 + M2_2);
    }
}

// measure distance in cm using sensor
int measuredistance(void)
{

    // setup timer to count the time ECHO is high
    TACCR0 = 0xFFFF;         // period
    TACTL = TASSEL_2 + MC_1; // TACLK = SMCLK, Up mode, start timer

    TACTL |= TACLR;          // clears the TAR value

    P1OUT |= TRIG;           // Start pulse
    __delay_cycles(10);      // Send pulse for 10 microsec.
    P1OUT &= ~TRIG;          // Stop pulse
    while(!(P1IN & ECHO));   // give time for pulse to trigger ECHO

    TAR = 0;                 // set timer = 0

    while(P1IN & ECHO);      // continue timer while ECHO is high

    sensor = (long)TAR;      // get data from TAR
    distance = sensor/56;    // convert data to distance in cm

    return distance;         // return distance measurement as an int type

}

// simulate back-up beeper using a PWM signal
void beep(void)
{
    CCTL1 = OUTMOD_7;              // CCR1 reset/set

    // beep twice
    for(int i = 0; i < 2; i++) {

        CCR0 = 1000-1;             // PWM Period
        CCR1 = 300;                // CCR1 PWM duty cycle
        TACTL = TASSEL_2 + MC_1;   // SMCLK, up mode
        __delay_cycles(400000);    // beep for 0.4s
        TACTL = TASSEL_2 + MC_0;
        CCR0 = 0;
        __delay_cycles(200000);    // pause for 0.2s

    }
}
```