EE318 Electronic Design Lab Project Report, EE Dept, IIT Bombay, April 2010

Ultrasonic 3-D Locator

Group No: 1

Ankita Sharma (07007002)
Chinmayee Shah (07007007)
Hutokshi Sethna (07d07025)

Supervisor: Prof. Dipankar

Abstract

The project aimed to design a device which can be used to locate the 3-d co-ordinates of a point in space. Ultra-sonic signals have been used for the purpose. The advantage of using ultrasonic signals is that more accuracy about the position can be obtained as against electromagnetic signals like infrared light or radio-frequency signals. This is because of the low speed of sound compared to light. Such a device can find application in locating an indoor device, 3d mouse, etc. Bard’s algorithm, which solves for the co-ordinate position by using time of difference arrival technique (TDOA) at the receivers, is used for the co-ordinate calculation.

1. Introduction

The device has two basic modules; location indicator and position calculator. The location indicator is the transmitter module which is independent of the calculation unit; which consists of receiver module and a processing unit. The transmitter module sends out a modulated pulse with carrier in ultrasound range. It is powered by the 10 V supply independent of the receivers.

The receiver module has 5 receivers located at pre-calculated positions so as to calculate the position of transmitter accurately. The procedure to calculate receiver positions and algorithm to estimate the position of the transmitter has been explained in detail in later section [2.2].

The processing module is basically a central microcontroller unit connected with all receivers which uses the conditioned signal from receiver module to calculate the exact time difference between receivers and thereby compute the location of the transmitter. The aim of the experiment is to get maximum accuracy possible with our method of detection i.e. ultrasound sensing. There is a prevalent method of using accelerometers in 3D mouse. But we tried this approach to detect locations which can be extended to even creation of 3D mouse as explained in section [5.2], since a continuous tracking of velocity and acceleration to calculate the position is not required in this case as in an accelerometer.

2. Design Approach

2.1. Hardware

The circuit consists of three major blocks:

1. The transmitter (along with the modulation circuitry)
2. The receiver (with the filters and amplifiers)
3. The central unit

The 3d co-ordinate of the object is detected based on the time difference of arrival at the receivers (explained in the algorithms section in detail). The transmitter sends pulses to the receivers, and the central unit calculates the delay between the receivers. Using these measurements, it then calculates the co-ordinate of the object (the transmitter).

For this purpose, we are using ultra-sonic transmitters and receivers. The reason for choosing ultra-sonic transmitters is that the speed of sound is approximately 340m/s. So, a path difference of the order of a few cm gives rise to a delay of a few micro-seconds, which can be easily measured to get the time difference. We have made use of 40 KHz ultra-sonic transmitter-receiver pairs for this device.

2.1.1. The transmitter module:

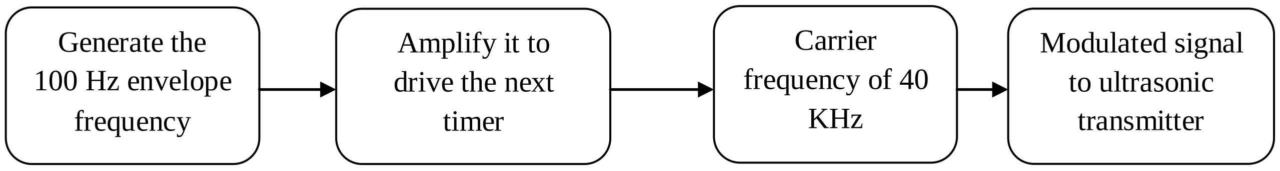

Generate the 100 Hz envelope frequency
Amplify it to drive the next timer
Carrier frequency of 40 KHz
Modulated signal to ultrasonic transmitter

Figure 1: Block Diagram of transmitter circuit

The ultrasonic transmitter-receiver pairs available in WEL Lab work at 40 KHz (with a tested bandwidth of approximately 2 KHz). We decided to use these transmitter-receiver pairs, since they were sufficient for our purpose. We do not need any particular frequency specific transmitters.

The bandwidth was tested by observing the frequency response at the receiver, as the frequency at the transmitter was varied using a frequency generator. A sharp deterioration of response is observed at 39 KHz and 41 KHz.

A frequency of 40 KHz implies a time-period of 25 µs. Taking the speed of sound in air at room temperature as 340 m/s for approximate initial calculations, a path difference of 1 cm gives a time difference of 29.4 µs. Thus, a difference of 1 cm in the path between the transmitter and two receivers would give rise to the two pulses at the receivers which are shifted by 29.4 µs with respect to each other. The expected differences in the path to two different receivers from the transmitter are greater than 1 cm. Since this corresponds to more than one cycle of 40 KHz, the path difference cannot be detected using simple phase measurements.

So, we modulated the carrier signal of 40 KHz with a 100 Hz signal with an extremely low duty cycle of approximately 5 per cent. This corresponds to a train of 20 pulses at 40 KHz followed by a blank phase, periodically at 100 Hz. The blank phase between two pulse trains lasts for 8 ms. A path difference of 1 m requires a blank phase of 3 ms, which is less than 8 ms upper limit.

This kind of modulation scheme is known as 'On-Off keying'.

2.1.2. The receiver module:

The circuit design consists of 5 receivers, which receive the transmitted signal, and the calculations are to be performed using TDOA (time difference of arrival) technique.

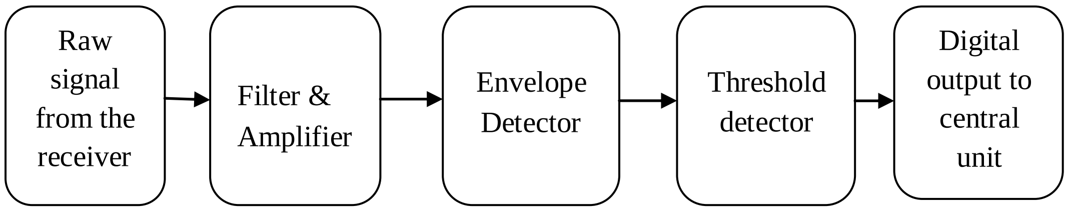


Figure 4: Block diagram of Receiver

The received signal is a 40 KHz carrier modulated by a 100Hz envelope. The raw signal received at the receiver (without any filtering or amplification), provided good power amplification is present at the transmitter, is of the order of 40 mV peak to peak, for a transmitter receiver distance of 20 cm. Converting this to acceptable logical levels implies an amplification of a factor of 125 is needed. This is where a major challenge lay for us. The received signal needed to be suitably amplified, with the low frequency noise filtered out. Envelope detection needed to be performed on this received signal to recover the 100 Hz message envelope. The signal thus obtained is then passed through a threshold detector to convert the signal to digital output in the form of 100Hz pulses which can be processed by the central unit to get the delay between the receivers.

Since the received signal is a 40 KHz square wave modulated by 100 Hz, the choice of amplifiers is dictated by the gain required. The circuit for amplification and filtering has been built using op-amps. We needed op-amps with a gain bandwidth product of at least 25 MHz since the gain required is approximately 125, and the signal being amplified is a square wave of 40 KHz (due to the square nature, a margin of a factor of 5 is multiplied, giving 200 KHz). A gain of 125 for a band of 200 KHz gives a minimum requirement of 25 MHz gain-bandwidth. The op-amps chosen have a bandwidth gain product of 63MHz.

2.1.3. The Central unit:

The central unit consists of a micro-controller and counters to calculate the delay between two received pulses. Atmega16 has been used because of the familiarity and ease with which it can be used. There are several open source tools available for Atmel micro-controllers, and we also have an usb-asp programmer for the same.

However, Atmega16 has only 3 counters and Atmega128 has only 4 counters. Out of the four counters available on Atmega128, two are 8 bit counters and two are 16 bit counters. We needed 5 counters for 5 receivers. We decided to operate the counters at 1 MHz clock since it would allow us to calculate the delay with a resolution of 1 μs. This corresponds to a path difference between transmitter and 2 different receivers of around 1mm. This could be expected for even large distances between transmitter and receiver, since though the length between the transmitter and receiver 1 and 2 might be huge, the difference between the two lengths can be very small, and the technique we are using essentially relies on the time difference. If we used 8 bit counter, then even a minimum resolution of 4 μs would allow a maximum value of 256x4, that is, 1024 μs to be calculated. This corresponds to only 34 cm. Thus, an 8 bit counter sets the upper limit for the range of the device at 34 cm, which is unacceptable. Hence, we needed to include external 16 bit counters in the central unit for delay calculation.

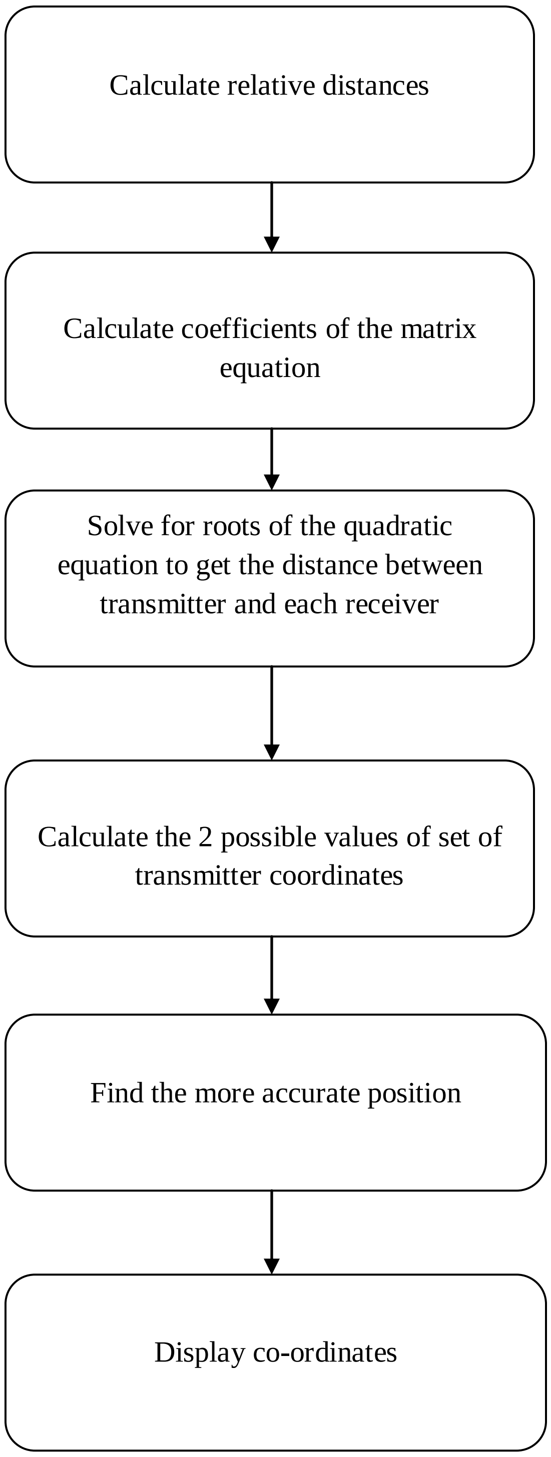
Intel offers counters 8253 and 8254, which can be operated with a clock of up to 2 MHz These are programmable counters, with interface available for micro-controllers and micro-processors. There are 3 counters of 16 bits in each chip, which can be operated in 5 different modes. Since these counters served our purpose, we decided to use these counters.

Our central unit consists of the receivers connected to an Atmega16, and another Atmgea16 controlling the two counters. The need for two micro-controllers rose because the pins available on one controller were not sufficient after adding the LCD and the five counters and receivers.

Since much synchronization was not needed between the 2 controllers, we decided to continue using Atmega16 instead of switching to a single Atmega128, which would have also served our purpose.

2.2. Algorithm for calculations and measurements, Software

Figure 5: Flowchart explaining Bard's Algorithm


Calculate relative distances
Calculate coefficients of the matrix equation
Solve for roots of the quadratic equation to get the distance between transmitter and each receiver
Calculate the 2 possible values of set of transmitter coordinates
Find the more accurate position
Display co-ordinates

The software for the project is divided into 2 parts. The first part deals with the counters used to find the time of arrival of the received pulses. We start each of the counters when the controlling receiver receives the pulse and stop all the counters once the last receiver has received the pulse signal. We then calculate the difference in the time of arrival of the signal at the receivers with respect to the reference receiver.

The second part is to calculate the 3-d co-ordinates of the transmitter using the difference in the time of arrival at the different receivers. The algorithm followed is the one proposed by John D. Bard.[1] The inputs to the algorithm are the position of the 5 receivers and the time difference of arrival of the signal at the receivers with respect to a reference receiver. Since the transmission is performed using ultrasound, the speed of sound in millimeters per second is used for calculations. This value is adjusted according to the measurements for the speed of sound made using an ultrasonic transmitter and a receiver. Typically, the speed of sound is found to be 333m/s.

Assuming propagation in a 3-d isotropic medium, the equation governing reception of the signal at the ith receiver is given by:

$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = c^2(\Delta t_i + t_0)^2$ [equation 1]

where,

c = speed of sound

(x_i, y_i, z_i) = co-ordinates of ith receiver

(x, y, z) = co-ordinates of transmitter

Δt_i = time difference of arrival at ith receiver and reference receiver

t_0 = time of arrival at reference receiver

The equation for reception at reference receiver is given by:

$x^2 + y^2 + z^2 = (ct_0)^2 = r_0^2 = |x|^2 = \mathbf{x}^\mathbf{T}\mathbf{x}$ [equation 2]

The equation relating time delays to the position of the receivers is

$a_i x + c\Delta t_i |x| = -1/2c^2 \Delta t_i^2 + ½|a|^2$ [equation 3]

where,

a_i is the position of the ith receiver.

The calculations yield 2 set of values for transmitter co-ordinates because we have used redundancies. This helps us to increase the accuracy because we can choose the best value of the calculated transmitter co-ordinates.

We have implemented the algorithm using matrix computations in C. The matrix of the receiver locations and the pseudo-inverse of the receiver locations are given to the function as inputs. The pseudo-inverse of the receiver matrix is calculated by us and fed to the program rather than calculating it within the code, to reduce the computations since the receivers are at a known constant location. The function is then implemented to check for the more accurate value of transmitter co-ordinates and would give only one set of co-ordinates as the result.

We performed a MATLAB simulation to test the algorithm.

3. Design Implementation

3.1. Hardware Design

3.1.1. Transmitter:

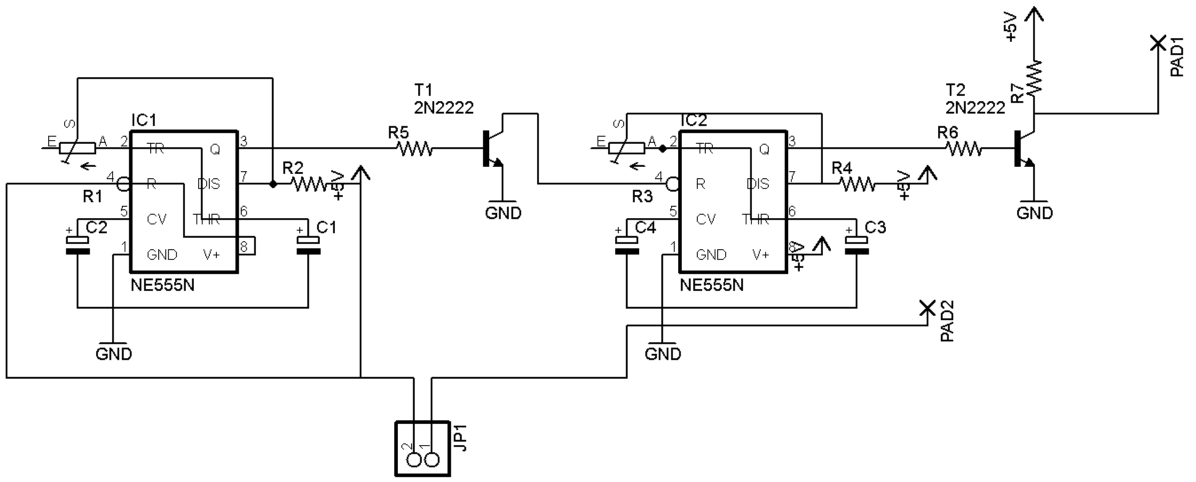


Figure 2: Transmitter Circuit Schematic

The modulation ‘on-off keying’, at the transmitter, has been implemented by driving the reset of one IC555 with the output of another IC555.

A 40 KHz square pulse (carrier) was generated using IC555 and this was applied to the transmitter after amplification using a BJT circuit. For the modulation, the reset of the 40 KHz IC555 is driven by inverted output of a 100 Hz IC555 operating at a duty cycle of 80 per cent. The inversion needed to be done because IC555 can produce a duty cycle of only 50 per cent or more in the astable mode. The inversion was done using a BJT.

The output of the 40 KHz IC555 is amplified using a BJT which drives the ultrasonic transmitter. We noticed that as the collector resistor is reduced, the power received at the receiver increases. This could be explained due to the increase in the charging current with a low resistor. Basically, the ultrasonic transmitter is a piezoelectric device with some capacitance. When the output of IC555 is low, the capacitor gets charged through the collector resistor. When the output of the IC555 goes high, the capacitor gets discharged through the BJT. As we reduced the collector resistor, the capacitor charging moved towards completion. Hence, a decrease in the resistance beyond a certain limit does not help in improving the transmitter power (once the capacitor is charged to around 90 per cent). This fact was verified by actual measurements at the receiver.

The value of the capacitance of the ultrasonic transmitter was calculated using the charging times observed. The calculated value was approximately 4 nF, which is close to the capacitance value of commercially available ultrasonic transmitters. (Datasheets of various other ultrasonic transmitters show capacitance value of 2.4 nF.)

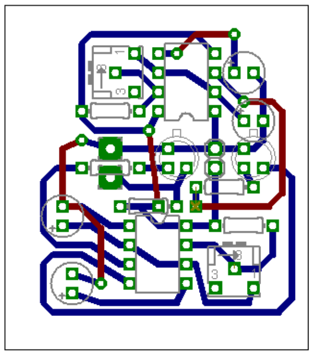


Figure 3: The PCB Layout for Transmitter

3.1.2. Receiver:

We initially built the amplifier circuit using LM324. However, the gain bandwidth product of LM324 is 1 MHz. The frequency of the received signal is 40 KHz, but since it is a square wave, a margin of 5 times 40 KHz, that is, 200 KHz needs to be kept. This allows an amplification of only 5. Initially we were not aware of these constraints and had designed the amplifier with a large gain. We were getting very poor frequency response for our circuit, as could be expected. The amplifier added trailing edges after the received signal, and the rise was also too slow.

So we changed the design to a single supply op-amp with greater gain bandwidth product. OP37 has a gain bandwidth product of 63 MHz. This would allow us an amplification of the order of $\frac{63000}{200}$, that is, 315. So, we decided to use OP37.

The amplification circuit consists of a non-inverting amplifier with dc bias at the input (since we are using single supply power supply, and the received signal is an ac signal). Initial measurements of the received signals had shown a large amount of noise at 100 Hz. This could be because of the A.C. power supply frequency of 50 Hz, which causes vibrations in equipments like air-conditioners. This was eliminated by choosing a 0.1 uF capacitor in series with the 1 K resistor at the input of the op-amp. The capacitor eliminates all low frequency noise. The resulting amplifier circuit is a high pass circuit with a cutoff at 1.59 KHz. Thus, the 100 Hz noise was eliminated, without affecting the 40 KHz received signal.

In the actual implementation, we needed to use two-stage amplification, as one-stage amplification was sufficient for short distances but proved to be insufficient for large distances. The amplification block is followed by an envelope detector. The envelope detector needs to filter out the 40 KHz carrier while retaining a reliable copy of the 100 Hz envelope signal modulating the carrier. A choice of 10 K resistor in parallel with 0.1 uF capacitor provides a cutoff at 160 Hz. The envelope detection was satisfactory with this circuit. A higher cutoff frequency (lower value of resistor or capacitor) gave 'peaky' output as the 40 KHz signal was not eliminated well enough.

The output of the envelope detector goes to a Schmitt trigger. A Schmitt trigger was used instead of an ordinary comparator to avoid any false triggering. With a certain threshold set, the output of the Schmitt trigger goes high when the received signal crosses a certain limit.

The initial amplification is followed by a subtractor circuit, which subtracts from the amplified signal, Vcc/2. This is because we only want to retain the part of the received signal which is greater than Vcc/2 (the received signal is symmetric about Vcc/2). The Schmitt trigger is preceded by an amplifier which acts as a buffer after the envelope detector, as also adjusts the voltage at the envelope detector by multiplying it by a factor of 2, since the output at the envelope detector needs to be multiplied by this factor before feeding it to the designed Schmitt trigger.

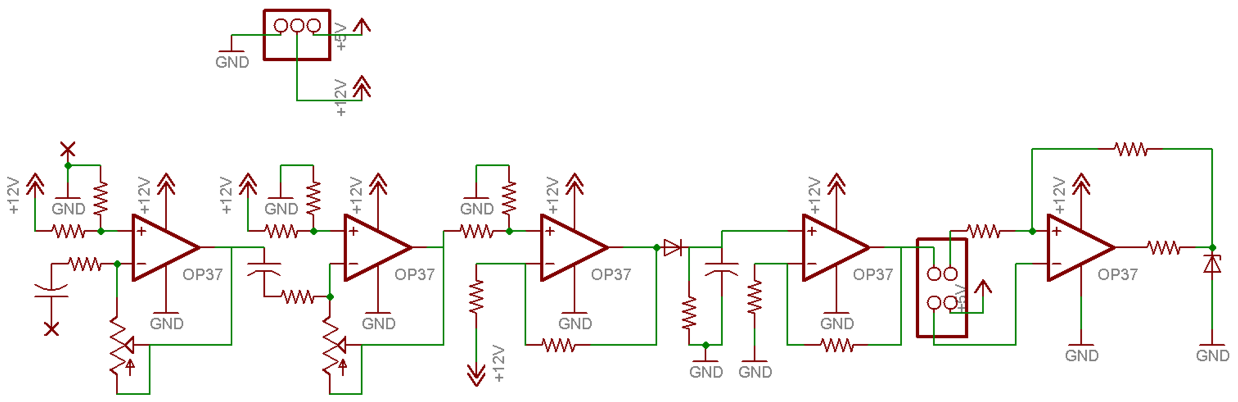


Figure 5: Schematic of receiver circuit

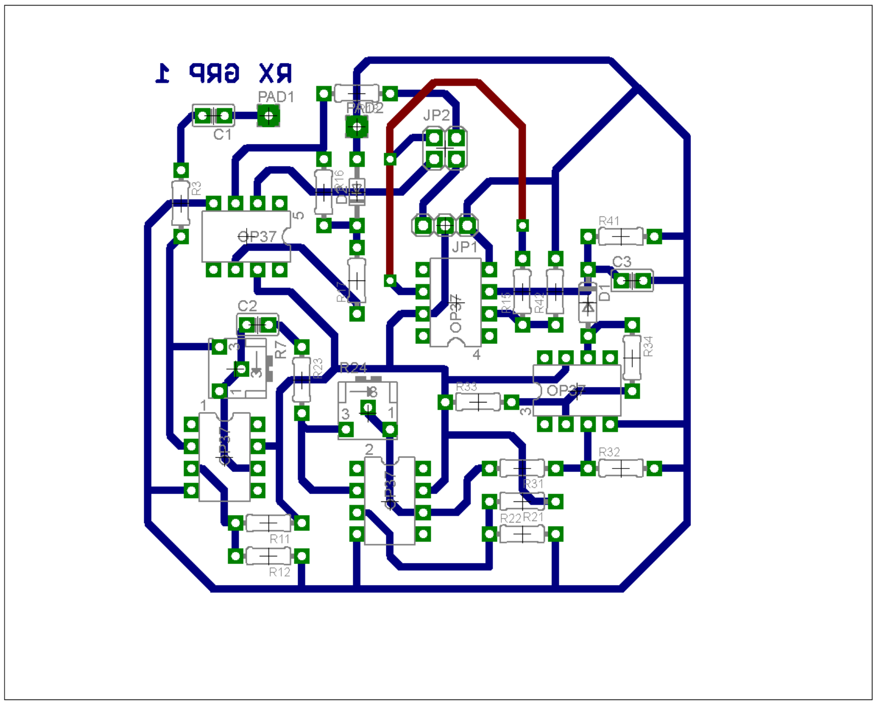


Figure 6: PCB layout of the receiver circuit

3.1.3. Central Unit:

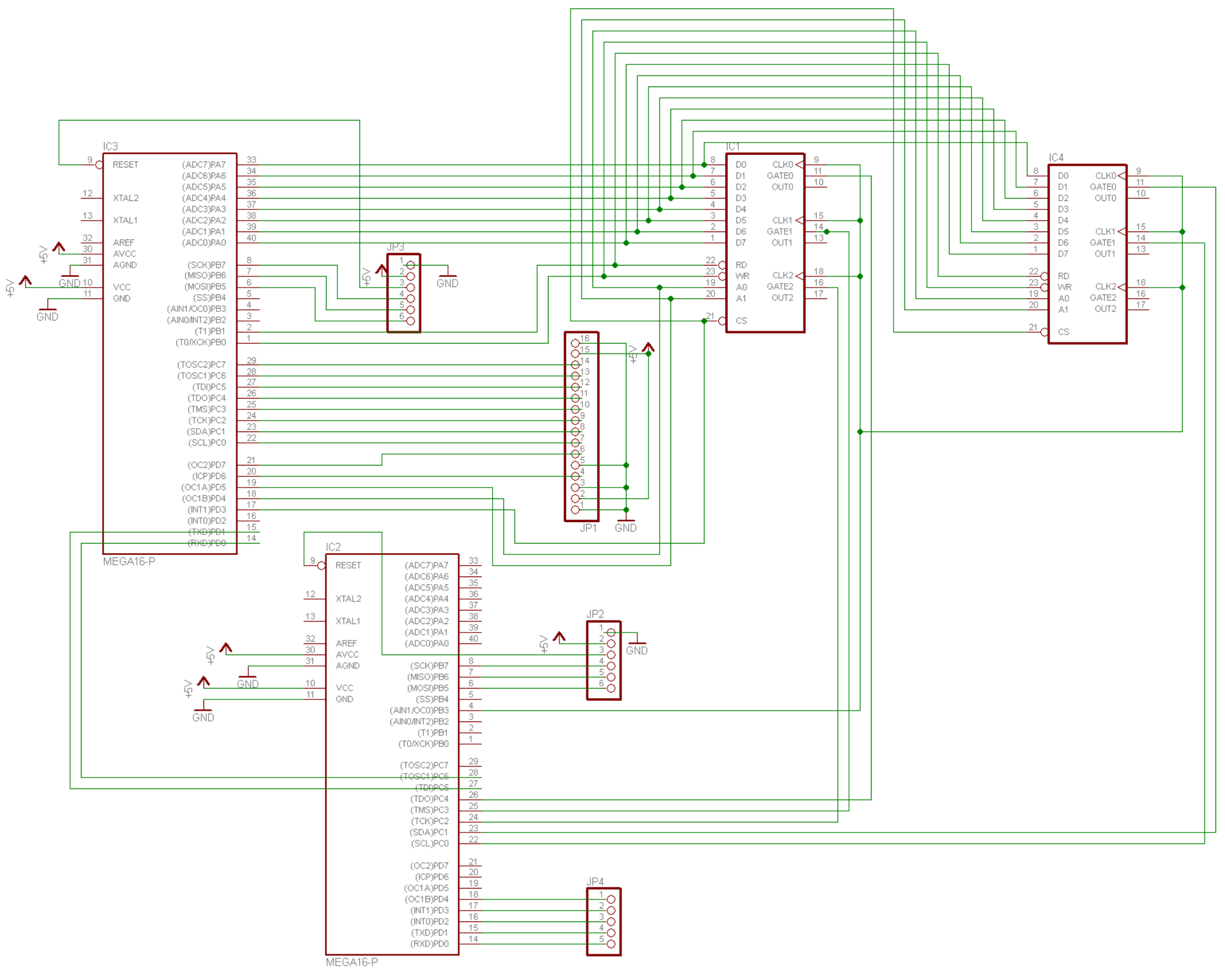


Figure 7: Schematic of central unit

The central unit consists of 2 Atmega16 micro-controllers and 2 8253 counters as explained before.

As receiver signals arrive, the micro-controller starts the corresponding counters and when the last signal arrives, all counters are stopped. The time difference can then be calculated with respect to a reference receiver. One controller monitors the receivers and starts the counters. The receivers and gates of the counters are connected to this micro-controller. It signals to the other controller when the counters have started and when the counting is over. Two connections have been made between the two micro-controllers for this purpose. The other controller then reads the counters, and performs the calculations, and displays the results on LCD.

3.2. Software

The software of the project is implemented with ATMega16 microcontrollers. The micro-controllers are used to control the counters in the IC8253. In the initialization function counter_init( ) we initialize the counter by setting the corresponding pins on the microcontroller IC and giving the counter a clock pulse of 1MHz using the timer0 of the microcontroller. The main code starts each counter when the respective receiver receives the signal pulse. When the last receiver receives the signal, all the counters are stopped. Then the readcount( ) function is called to read the value of counters. The counter is decremented from FFFF so the value of the count corresponding to the time is the difference of the counter value from FFFF. Once the counter values are read, the setcount( ) function is called to reset the values of the counter. After we get the counter values we find the time difference with respect to the reference receiver using the function findtime( ). Then we give this array of time differences to the bard solver function – bard_solver( ) as input along with the receiver matrix and the pseudo-inverse of the receiver matrix. The calculated co-ordinates of the transmitter are then displayed on the LCD display.

4. Problems, Testing procedure and Results

Final status and Problems:

- We have implemented the transmitters and receivers and the central unit.
- The transmitter receiver circuit is not very robust to noise. Initially we had noticed noise of around 50 Hz from electrical equipments and had added a high pass filter to remove this noise.
- However recently, some high frequency noise was noticed occasionally, which dominated the transmitter signal. As a result, the receiver signal gets corrupt with no reliable data. In such circumstances, no results are produced.
- The transmitter has a very small cone angle. This actually limits the range of the device since the receivers need to be placed close to each other for good detection of the transmitted signal. Placing the receivers close to each other reduces the range of the device and results in less accuracy.

Testing procedure and Results:

Testing of the final device, once assembled, is as follows:

- The output of the individual IC555 and the output of the BJT transistor driving the transmitter can be checked. This ensures the working of the transmitter driving circuitry. An ultrasonic receiver can be placed in front of the transmitter and its raw signal can be checked to ensure that the ultrasonic transmitter itself is working.

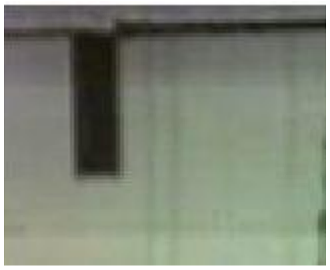


Figure 8: The transmitted (modulated) waveform. The carrier wave of 40 KHz is not visible because of higher time scale in CRO. But the basic shape of pulse can be seen here.

- The raw signal at the receiver is as shown below:

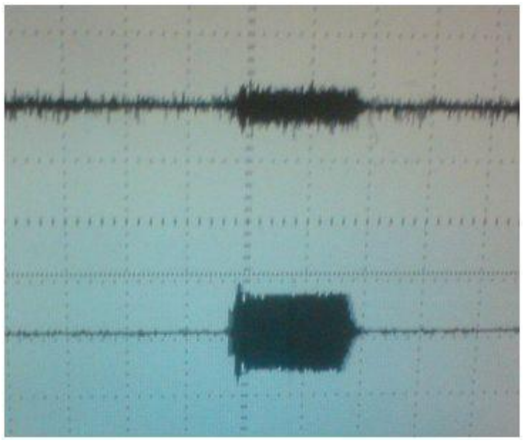


Figure 9: the image shows raw received signal at the receiver (signal is the transmitted signal shown in figure 8) and below it is the amplified version to required scale.

The above signal at the receiver ensures that the transmitter and the receivers pairs are working (and are not damaged).

- The final transmitter output after amplification and envelope detection is as shown:

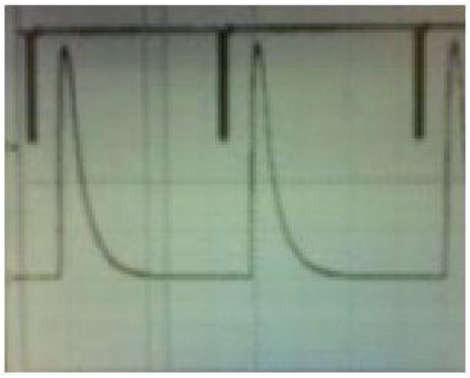


Figure 10: the image shows the transmitted signal and the output of the envelope detector of the receiver module.

This ensures the working of the amplification, filter and threshold detector circuits.

- A dummy code can be burnt to the two micro-controllers to ensure that the central unit, which includes the two micro-controllers and counters are working and the connections are correct.

- The time difference between the different receivers, and the corresponding path lengths can be displayed, along with the calculated co-ordinate, to check if the delay measurements and path calculations have been made correctly.

- We have also made a MATLAB simulation to check the algorithm that we have implemented. Given the position of the receivers, once the transmitter co-ordinates are fed, the simulation can give the delays at the receivers, and given the delays, the position of the transmitter.

- The MATLAB simulation helps to check how close to theoretical expectations the actual results are.

Some small experiments had been performed while developing the hardware. The results have been summarized below.

Table 1: Initial measurements of the amplitude vs. distance for the receiver

Sr no	Distance	Voltage at receiver
1	10 cm	80 mV
2	20 cm	50 mV
3	30 cm	40 mV
4	40 cm	20 mV

From these measurements, we concluded that we would need to amplify a voltage of the order of 20 mV, and arrived at the gain factor of 125.

Table 2: Measurements of time difference

Sr no	Distance	Time difference
1	2 cm	64 µs
2	4 cm	150 µs
3	6 cm	208 µs
4	8 cm	268 µs

From the data, it can be concluded that a path difference of 2 cm gives rise to a delay of approximately 60 µs, which gives the speed of sound as 333 m/s. The speed of sound calculated in this manner will help to calibrate the final device, since we need to calculate the path difference from the time difference. The measurements also depict that the delay obtained is quite reasonable and reliable for the final calculation of the path lengths and the co-ordinate of the transmitter.

6. Future work and improvements

- The transmitter receiver circuits can be made robust to the high frequency noise by adding a band pass filter instead of a high pass filter which eliminates only the low frequency noise.
- The range and accuracy can be improved by using omni-directional transmitter, or a transmitter with a wider beam. Another solution is to design a good transmitter array. However, care needs to be taken that voids with low transmitter signal are not left in between.
- ‘Auto-calibration’ can be implemented for varying atmospheric conditions. The speed of sound can automatically be calculated by first placing the receivers at a pre-decided location. The range of the device can then be decided by placing the transmitter at the corners of an imaginary cube. The accuracy of detection will then depend on the receivers.

6. Conclusions

From the above results and observations that we have compiled of our device, we can draw following conclusions:

- It is a low power position locator working on ultrasonic transmitted pulses and the received signal is processed to get the position of the transmitter.
- The device can detect position of transmitter wherein the accuracy depends on the processing unit (bit resolution of the time differences) and the transmitter beam angle.
- The device can be made more robust with auto-calibration.
- The device can be made more user-friendly. Its use can be extended to detection of an object in a room or a mouse.

7. References

[1] J. D. Bard, F. M. Ham, and W. L. Jones. An algebraic solution to the time difference of arrival equations. South east con'96. 'Bringing Together Education, Science and Technology'. Proceedings of the IEEE, pages 313–319, 1996

[2] "Transceiver Circuits"
http://www.ecelab.com/circuit-ultrasonic-t.htm , http://www.ecelab.com/circuit-ultrasonic-r.htm

[3] Atmega16 datasheet
http://www.atmel.com/dyn/resources/prod_documents/doc2466.pdf

[4] 8253 and OP27, OP37 datasheet
http://www.datasheetcatalog.com

[5] Other datasheets
http://wel.ee.iitb.ac.in/wel12/resources.html

8. Appendix

8.1 List of Major Components Used:

- 1 ultrasonic transmitter & 5 ultrasonic receivers
- Atmega 16
- 8254 counter
- OP37
- OP27
- LM555
- Resistors, capacitors, potentiometers, diodes

8.2. Software Codes

Code for the micro-controller which receives the signals from the receivers and starts the counters:

```
#include <avr/io.h>
#include <util/delay.h>

#define GATE        PORTC
#define GATE_DDR    DDRC
#define GATE0       PC4
#define GATE1       PC3
#define GATE2       PC2
#define GATE3       PC1
#define GATE4       PC0


#define RX_DDR      DDRD
#define RX0     PD4
#define RX1     PD3
#define RX2     PD2
#define RX3     PD1
#define RX4     PD0
#define CLK     PB3
#define RX      PIND

#define SIGNAL1     PC5
#define SIGNAL2     PC6
```

```
int count, r0, r1, r2, r3, r4;
void timer0_init(void)                                                          // Initialize timer
{
        TCCR0 = _BV(WGM01)|_BV(COM00)|_BV(CS00);
        OCR0 = 0x03;
        DDRB |= _BV(PB3);
}

void main()
{
timer0_init();
char state = 0;
GATE_DDR|=_BV(GATE0)|_BV(GATE1)|_BV(GATE2)|_BV(GATE3)|_BV(GATE4)|_BV(SIGNAL1);
GATE_DDR &= ~_BV(SIGNAL2);
//set SIGNAL2 to INPUT
RX_DDR &= ~_BV(RX0) & ~_BV(RX1) & ~_BV(RX2) & ~_BV(RX3) & ~_BV(RX4);
// Receivers as INPUT
GATE &= ~(_BV(SIGNAL1));        //Set SIGNAL1 to 0
DDRB = DDRB | _BV(PB5) | _BV(PB6) | _BV(PB7);
PORTB &= ~(_BV(PB5));


while ( state == 0 )
{
        GATE &= ~(_BV(SIGNAL1));        //Set SIGNAL1 to 0
        char notfirst = 0;
        //wait while even one of the receivers is low

        while (notfirst == 0)
        {         //to get reliable results
                _delay_us(10);
                 if (((RX & _BV(RX0)) && (RX & _BV(RX1)) ))
                {
                        _delay_us(10);
                        notfirst = 1;
                         if (((RX & _BV(RX0)) && (RX & _BV(RX1)) ))
                        notfirst = 1;
                }
        }

        state = 1;
        count = 0;
        r0 = 0;
        r1 = 0;
        r2 = 0;
        r3 = 0;
        r4 = 0;

        while (count < 5)                                                    {
        if(!((RX & _BV(RX0)) || r0))    //when rx0 goes low, and for the first time
        {
                GATE |= _BV(GATE0);               //start counter for rx0
                count = count + 1;
                r0 = 1;
        }

        if(!((RX & _BV(RX1)) || r1))
        {
              GATE |= _BV(GATE1);
              count = count + 1;
              r1 = 1;
        }

        if(!((RX & _BV(RX2)) || r2))
        {
              GATE |= _BV(GATE2);
              count = count + 1;
              r2 = 1;
        }

        if(!((RX & _BV(RX3)) || r3))
        {
              GATE |= _BV(GATE3);
              count = count + 1;
              r3 = 1;
        }

        if(!((RX & _BV(RX4)) || r4))
        {
              GATE |= _BV(GATE4);
              count = count + 1;
              r4 = 1;
        }
 }
//stop all counters
GATE &= ~_BV(GATE0) & ~_BV(GATE1) ;//& ~_BV(GATE2) & ~_BV(GATE3) & ~_BV(GATE4);

//give a 10usec pulse on SIGNAL1
```

```
                        GATE |= _BV(SIGNAL1);
                        _delay_ms(1);
                        GATE &= (~_BV(SIGNAL1));

                        _delay_ms(6);   //wait to avoid reflections diffusions etc


                        state = 0;
            }
}
```

Code for the micro-controller handling the display and bard solver:

```
#include <avr/io.h>
#include<util/delay.h>
#include <math.h>
#include <counter.c>
#include <lcdroutines.c>
#include "lcdroutines.h"
#include <bard_solver1.c>
#include <bard_solver2.c>

float t_delta[4];
int count0=0,count1 = 0,count2 = 0,count3 = 0, count4 = 0;
int delaym[4];
int timedifffirst[3];
int timediffsecond[3];
float cord1[3];
float cord2[3];
int cord[3];

void display()
{
            unsigned char message[] = {"time diff"};
            for(unsigned char i =0;i<sizeof(message)-1;i++)
            display_char(message[i]);
            _delay_ms(2000);
            lcd_clear();

            unsigned char space[] = {"  :  "};

            display_int(1);
```

```
for(unsigned char i =0;i<sizeof(space)-1;i++)
display_char(space[i]);
display_int(delaym[0]);
_delay_ms(1000);
lcd_clear();

display_int(2);
for(unsigned char i =0;i<sizeof(space)-1;i++)
display_char(space[i]);
display_int(delaym[1]);
_delay_ms(1000);
lcd_clear();

display_int(3);
for(unsigned char i =0;i<sizeof(space)-1;i++)
display_char(space[i]);
display_int(delaym[2]);
_delay_ms(1000);
lcd_clear();

display_int(4);
for(unsigned char i =0;i<sizeof(space)-1;i++)
display_char(space[i]);
display_int(delaym[3]);
_delay_ms(1000);
lcd_clear();

unsigned char message1[] = {"co-ordinates"};
for(unsigned char i =0;i<sizeof(message1)-1;i++)
display_char(message1[i]);
_delay_ms(2000);
lcd_clear();

display_char('x');
display_char(':');
display_int(cord[0]);

move_to(8, 0);
display_char('y');
display_char(':');
display_int(cord[1]);

move_to(0,1);
display_char('z');
```

```
        display_char(':');
        display_int(cord[2]);

        _delay_ms(2000);
        lcd_clear();

}

void timediff()
{
        count0 = (255 - count0h)*256 + (255 - count0l);
        count1 = (255 - count1h)*256 + (255 - count1l);
        count2 = (255 - count2h)*256 + (255 - count2l);
        count3 = (255 - count3h)*256 + (255 - count3l);
        count4 = (255 - count4h)*256 + (255 - count4l);
        t_delta[0] = count1 - count0;
        t_delta[1] = count2 - count0;
        t_delta[2] = count3 - count0;
        t_delta[3] = count4 - count0;
        delaym[0] = t_delta[0];
        delaym[1] = t_delta[1];
        delaym[2] = t_delta[2];
        delaym[3] = t_delta[3];
        if (delaym[0] < 0)
        delaym[0] = -delaym[0];
        if (delaym[1] < 0)
        delaym[1] = -delaym[1];
        if (delaym[2] < 0)
        delaym[2] = -delaym[2];
        if (delaym[3] < 0)
        delaym[3] = -delaym[3];

        timedifffirst[0] = t_delta[0];
        timedifffirst[1] = t_delta[1];
        timedifffirst[2] = t_delta[2];

        timediffsecond[0] = t_delta[0];
        timediffsecond[1] = t_delta[1];
        timediffsecond[2] = t_delta[3];
}


void main()
{
	lcd_init();

	unsigned char first[] = {"first"};
	for(unsigned char i =0;i<sizeof(first)-1;i++)
	display_char(first[i]);
	_delay_ms(2000);
	lcd_clear();

	CONTROL_DDR &= ~_BV(SIGNAL1);
	CONTROL_DDR |= _BV(SIGNAL2);
	_delay_ms(50);

	counterinit();

	while(1)
	{
		if (SIGNAL_RX & _BV(SIGNAL1))
		{
			CONTROL_PORT |= _BV(SIGNAL2);
			readcount();
			setcount();
			timediff();

			bard_solver1(timedifffirst, cord1);
			bard_solver2(timediffsecond, cord2);
			for (int k = 0; k<3; k++)
			cord[k] = (cord1[k] + cord2[k])/2;
			display();
			_delay_us(10);
			CONTROL_PORT &= ~_BV(SIGNAL2);
		}
	}
}
```

One of the bard solver codes in C:

```
#include <math.h>
//inputs required : receiver matrix and pseudoinverse, time of arrival at each receiver

void bard_solver1(int timediff1[3], float cord_out1[3])
{
    /////////////////////////////////Initializations////////////////////////////////////////
```

```
float A[3][3];
float invA[3][3];
const float c = 34000;
float m[3];
float  AAt_diag[3]={0,0,0};
float phi[3][3];
float c_delta[3];
float aa_sum[3] = {0,0,0};
float bb_sum[3]= {0,0,0};
float aa,bb,cc;
float root;
float sL[2];
float s1[3][2];
float s2[3][2];
float xl[3][2];

///////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////      Receiver Matrix          ////////////////////////////

                    A[0][0]=27;
                    A[0][1]=0;
                    A[0][2]=0;
                    A[1][0]=15;
                    A[1][1]=-25;
                    A[1][2]=0;
                    A[2][0]=-8;
                    A[2][1]=23;
                    A[2][2]=25;


         //get matrix from include file

//////////////////////////      Inverse Matrix          /////////////////////////////

                    invA[0][0] = 0.0370;
                    invA[0][1] = 0;
                    invA[0][2] = 0;
                    invA[1][0] = 0.0222;
                    invA[1][1] = -0.04;
                    invA[1][2] = 0;
                    invA[2][0] = -0.0086;
                    invA[2][1] = 0.0368;
                    invA[2][2] = 0.04;
```

```
 ///////////////////////////    Calculating Constants        ///////////////////////////

 ////////////////               m = 1/2 diag( AA' - c^2(delta_t)(delta_t)')      /////////////////
  for (int i = 0;i<4;i++)
  {
     c_delta[i] = c*((float)(timediff1[i]))/(1000000);
     for(int k=0;k<3;k++)
     {
         AAt_diag[i]+= A[i][k] * A[i][k];

     }
     m[i] = 0.5 * (AAt_diag[i] - c_delta[i]*c_delta[i]);

////////////////////////        phi = inv(AA')           /////////////////////////////////////

                    phi[0][0]=0.0019;
                    phi[0][1]=-0.0012;
                    phi[0][2]=-0.0003;
                    phi[1][0]=-0.0012;
                    phi[1][1]=0.0030;
                    phi[1][2]=0.0015;
                    phi[2][0]=-0.0003;
                    phi[2][1]=0.0015;
                    phi[2][2]=0.0016;
          }

 ///////////////////     Calculating co-eff of quadratic   /////////////////////////////////

  for(int i = 0;i<3;i++)
  {                 aa_sum[i]=c_delta[0]*phi[0][i]+c_delta[1]*phi[1][i]+c_delta[2]*phi[2][i];
        bb_sum[i]=m[0]*phi[0][i]+m[1]*phi[1][i]+m[2]*phi[2][i};
  }
        aa=(aa_sum[0]*c_delta[0]+aa_sum[1]*c_delta[1]+aa_sum[2]*c_delta[2])1
        bb=bb_sum[0]*c_delta[0]+bb_sum[1]*c_delta[1]+bb_sum[2]*c_delta[2    bb=bb*-2;
        cc=bb_sum[0]*m[0]+bb_sum[1]*m[1]+bb_sum[2]*m[2];//9+bb_sum[3]*m[3];


  ////////////////////      calculating s = ||x||        //////////////////////////////////

 root= bb*bb - 4*aa*cc;
if(root>0)
root=sqrt(root);
else
root=0;

sL[0]=((-bb+root)/(2*aa));
sL[1]=((-bb-root)/(2*aa));

   ///////////////////      calculating co-ordinates     ///////////////////////////////

         for(int i = 0;i<3;i++)
         {
                  for(int j = 0;j<2;j++)
                  {
                           s1[i][j]=sL[j]*c_delta[i];
                           s2[i][j]=m[i]-s1[i][j];
                  }

         }



   for(int i = 0;i<3;i++)
         {
                  for(int j = 0;j<2;j++)
                  {
                           xl[i][j]= (invA[i][0]*s2[0][j]+invA[i][1]*s2[1][j]+invA[i][2]*s2[2][j])/3;
                  }
         }

         ///////////////////     find more accurate solution     ///////////////////////////


         cord_out1[0]=xl[0][1];
         cord_out1[1]=xl[1][1];
         cord_out1[2]=xl[2][1];


}
```

The counter code for 8253 and MATLAB simulation are not included.