Julius Gachiri Wamweya 53804100

HC-SRO4 DOPPLER VELOCITY METER

ABSTRACT

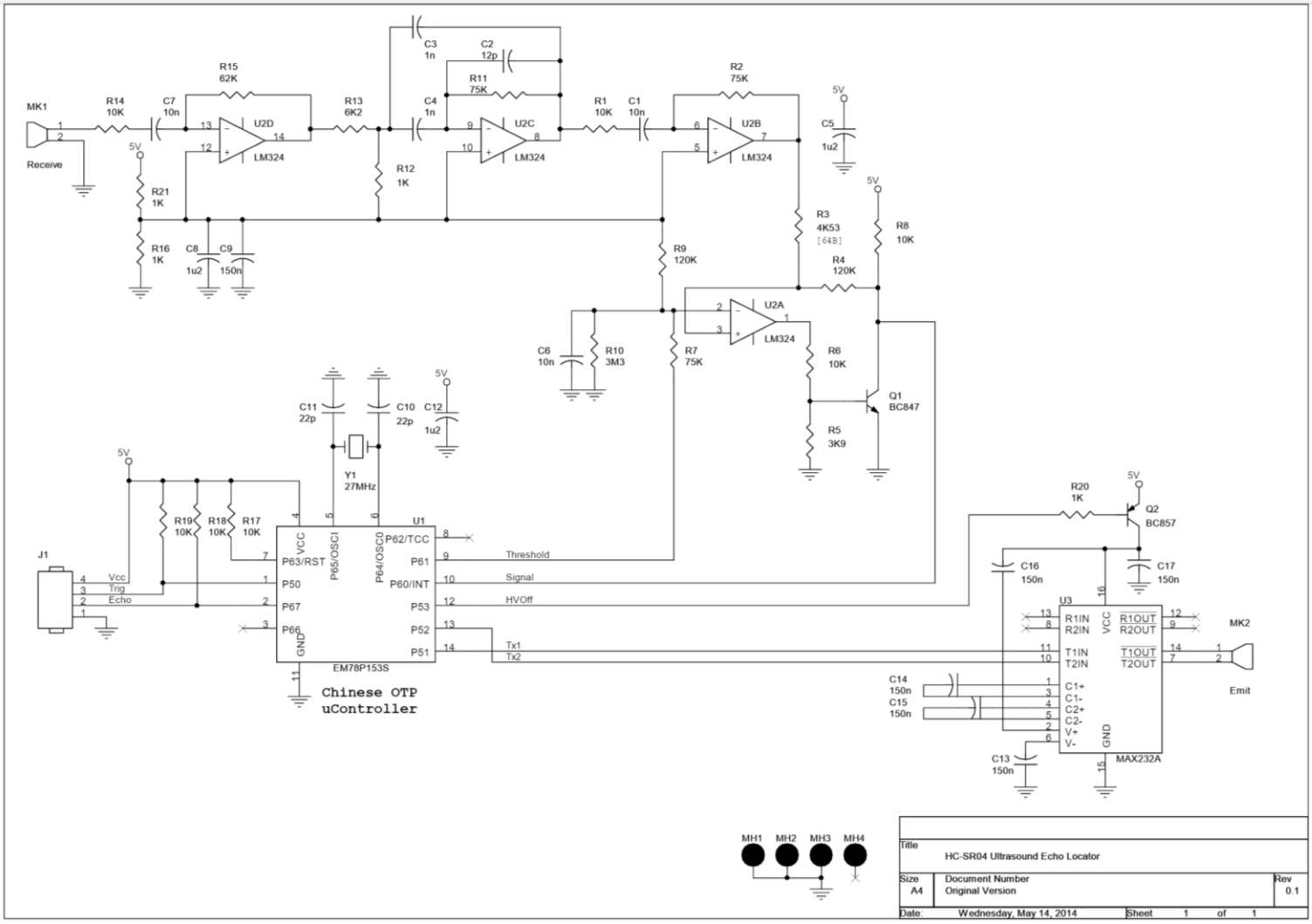
The objective was to test the capacity of a modified HC-SRO4 Ultrasonic distance sensor to function as a Doppler velocity meter that is driven by an MSP430 microcontroller. The sensors' main controller was removed while the transducers that emit the sound waves and the receivers that listen for the echo were repurposed by adding an exclusive or-gate to detect any phase difference. The MSP430 microcontroller ran flashed assembly code that instructed the on-chip pulse width modulator to emit the initial wave and converted phase difference with the echo into a velocity as in the Doppler Effect formula. Repeated measurements were transmitted to a host computer via USB for graphing and storage via a Python program. The velocity meter was unreliable, highly susceptible to noise and only functional within a small range of 2 – 20 cm with large errors of up to ± 5cm/s. The HC-SCR04 proved a bad candidate for sensors in this particular application.

INTRODUCTION

The Doppler Effect, first proposed by Christian Doppler in 1842, is observed when a travelling wave from a source has an apparent change in frequency from the perspective of an observer in relative motion[1]. His hypothesis was first confirmed and is most commonly observed when sound waves are emitted from a moving source such as an ambulance's siren. The sound pitch raises and lowers as the source approaches and recedes respectively. This observed change in frequency can be used to calculate the relative velocity of the source. The phenomenon has since found a wide variety of practical applications including in medical imaging, non-invasive fluid velocity measurement, Doppler radars, satellite communication and in exoplanet discovery and confirmation via Doppler radial velocity[2]. Edwin Hubble famously postulated an expanding universe due to the observed decrease in expected frequency (red-shifting) of objects in deep space[3]. In our final lab we used an Ultrasonic sensor to measure distances. My main motivation was to test the capability of the HC-SRO4 to detect and measure relative velocity via the Doppler Effect.

THEORY

The key component of the project is the HC-SRO4.



HC-SRO4 Circuit[4]

It has 4 pins - V_{cc} (5V), ground, an input trigger and echo output. Measurements are processed by the Chinese OTP uController that drives a MAX232A driver with a transducer (MK2) for signal emission. The receiver circuit consists of a second transducer (MK1) connected to a times 6 amplifier, a multiple feed-back band filter, a times 8 amplifier and a hysteresis comparator. If the trigger input is raised for 10 µs, the uController sets the comparator threshold high to prevent the receiver from picking up any signal directly from the sending transducer. It then turns on MAX232A, powering MK2 which produces an 8 pulse 40 kHz signal. After emission, MAX232A's power is cut off and the comparator threshold is lowered. The echo is detected by MK1, amplified by U2D, filtered, further amplified by U2B and the comparator with Q1 reduces the noise. The uController receives this signal and raises the echo output pin for a period proportional to the time between sending the signal and receiving the echo. Measuring the width of this pulse and factoring in speed of sound in air results in a distance measurement[5]. For Doppler velocity measurements, the idea is to bypass the uController using the MSP430 micro-controller for sending the initial pulse which is also continually passed to an XOR phase detector as the base line signal for comparison with the received echo. Any phase difference is then directly proportional to the duty cycle of the phase detector output signal. If the emitted signal reflects off of a hard surface in relative motion to

the HC-SRO4, then the Doppler shifted echo signal produces a phase difference from the phase detector which is received by the MSP430. Twice the average time difference between successive peaks of the phase difference signal is ΔT.

$$\frac{1}{\Delta T} = \Delta F$$

$$\Delta V = \text{Speed of Sound in Air} \times \frac{\text{Transmitted Signal Frequency}}{\Delta F}$$

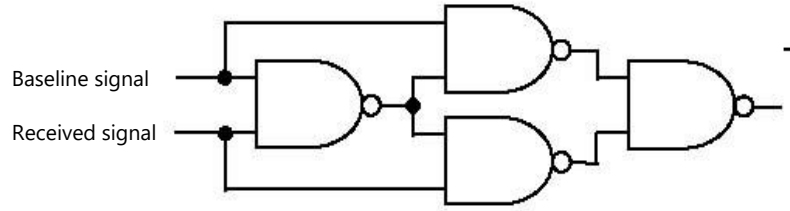
Doppler velocity calculation[6]

Since the device is to be kept at rest, the calculation above from the Doppler Effect returns the velocity. It is to be transmitted one bit at a time via the MSP430's UART to the host computer for plotting and storage. Repeated measurements are to be taken to get velocity over a time span.

APPARATUS

After removing the uController, connecters were soldered to P61, P60, P53, P52 and P51 (as in the HCR-S04 circuit diagram) on the EM78P153S socket allowing MAX232A and the comparator threshold to be driven and the echo signal to be received directly. I connected the comparator threshold (P61) to P2.3 on the micro-controller, MAX232A power supply (P53) to P2.2, sending transducer inputs (P52 and P51) to the pulse width modulator on P2.1 and the received signal (P60) as one of the inputs to the phase detector. I constructed the X-OR phase detector using 4 NAND gates due to unavailability of its chip.

Exclusive OR (XOR) **Phase detector**[7]



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

The base-line signal from the MSP430 pulse width modulator and the received echo were input to the phase detector. The output was connected to P1.2 – timer0_A capture compare input pin. The MAX232A is turned on and off with inputs from the MSP430 P2.2. The program that runs the MSP430 is main.c (appendix 1), a modified form of the temperature measurement program. To allow for time to detect frequency changes, the MSP430 internal clock is calibrated to 16 MHz. A pre-application mode toggles the green and red LEDs awaiting a press of button S2 that causes an interrupt that launches the main application. In main the pulse width modulator is set by timer A1 control register with a 3 times divider to generate a 40 kHz signal from a 50 clock tick period with a 50% duty cycle. This signal output is continuously used as the baseline input for the XOR phase detector and also in bursts as the ultrasonic sensors' input. The MAX232 on/off pin is initially set to high and the comparator threshold pin is also set as high for a wait count of 400 clock ticks. This allows a pulse train that is emitted from the sending transducer and not picked up by the receiver directly. The MAX232A is subsequently shutoff (P2.2 low) and the comparator threshold set to low – listening for an echo whose frequency difference would be

output by the XOR phase detector. The MSP430 enables interrupts and goes into low power mode 0 with interrupts enabled on timer_A0 capture compare control register for any rising or falling edges and interrupts on timer_A0 overflows. If the phase detector outputs any frequency change, switches from low to high or high to low in this signal cause timerA0 interrupts that are captured as time recordings by the timer_A0 capture compare register and stored in the array Times[4]. The times difference between the duration of two successive high intervals is given by: $\frac{\Delta T}{2} = |(\text{Times}[4] - \text{Times}[3]) - (\text{Times}[2] - \text{Times}[1])|$. ΔF ($\Delta F = \frac{1}{\Delta T}$)is multiplied by the speed of sound in air (34320 cm/s) and divided by the emitted frequency (40 kHz) to give a velocity reading as in the Doppler velocity formula[6]. Since time is measured in timer_ A ticks at 2 MHz, for a velocity in cm/s a factor of 1000*2000000/ 1166 is used.

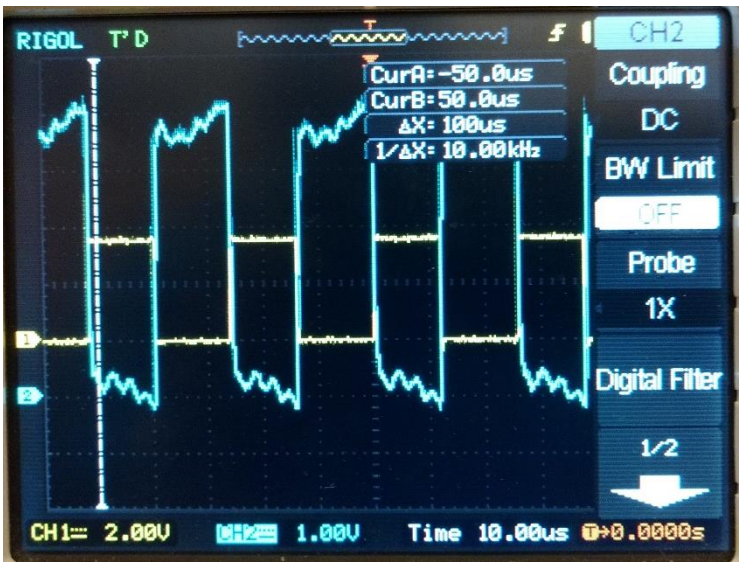
$$v = \frac{34320\frac{\text{cm}}{\text{s}} \times \Delta F}{40\ \text{kHz}} = \frac{34320\frac{\text{cm}}{\text{s}}}{40\ \text{kHz} \times \Delta T}$$

$$v = \frac{34320\frac{\text{cm}}{\text{s}} \times 2\text{MHz}}{40\ \text{kHz} \times \text{delta\_t}(\text{in timer}_A\text{ticks})} = \frac{1000 \times 2\text{MHz}}{1116 \times \text{delta\_t}(\text{in timer}_A\text{ticks})}$$

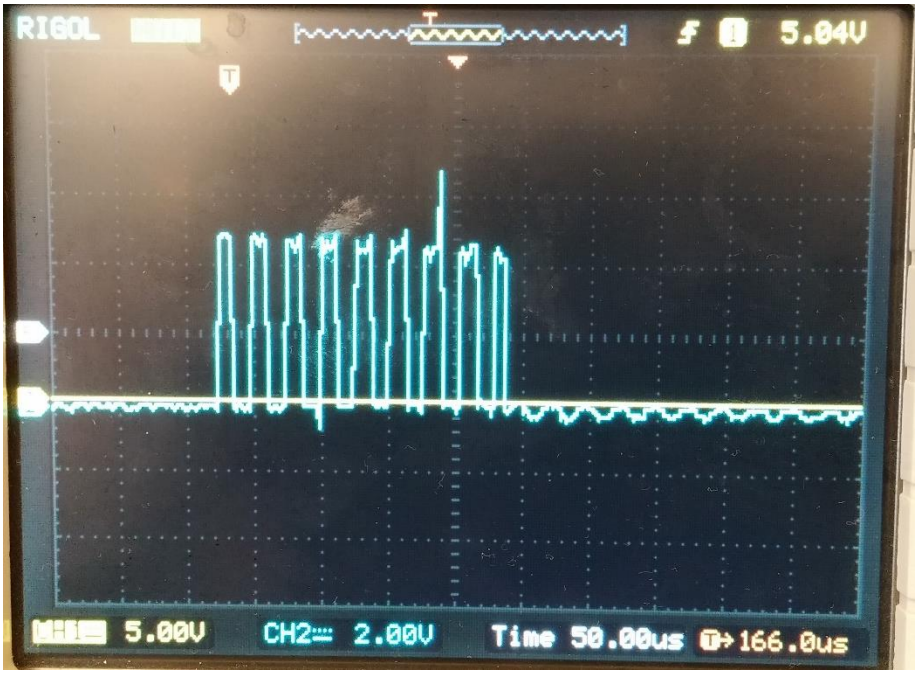
The MSP430 awakes from lower power mode 1 ms after taking a measurement or if timer A0 overflows 4 times (13.1 ms) in the case of no change in frequency output from the phase detector (velocity = 0). The velocity is set as the TXByte and the UART is configured by selecting alternate functions for the TXD (P2.1) and RXD (P2.2) pins. The TXByte is transmitted 1 bit at a time with a 2400 baud rate translating to an 833 clock tick bit time. After transmission, the red LED is toggled. The velocity is then plotted on a velocity vs time curve on the host computer using the same python program (appendix 2) for measuring temperature in lab 5_6 and stored in time_and_velocity.txt as a reading with time measurement. The main application loops after a 20.48 ms delay to allow the MAX232A time to recharge. This produces a continuous velocity measurement and runs until the MSP430 is shutoff or reset.

RESULTS

The signal seen below was generated by the pulse width modulator and continually fed to the phase detector as the baseline signal as required (40 kHz).

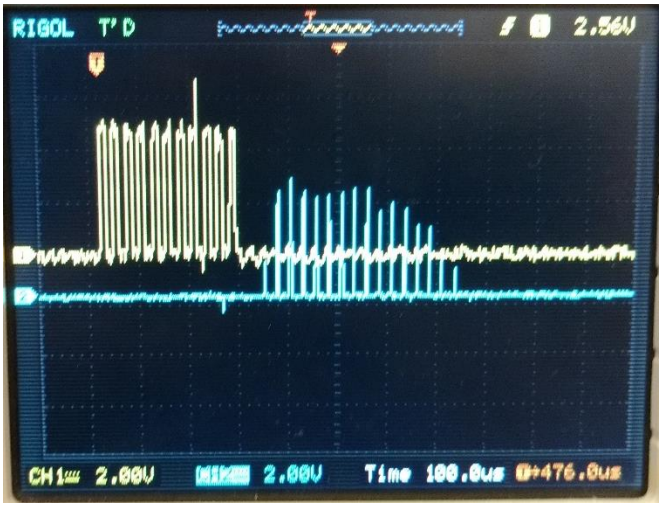


On each loop of the main application, the pulse train below is sent to and emitted from the MK1 transducer.

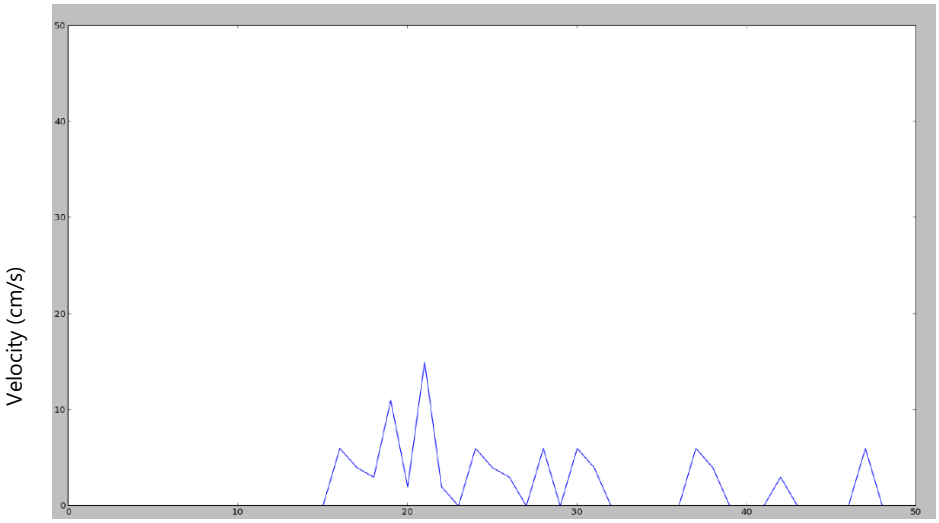


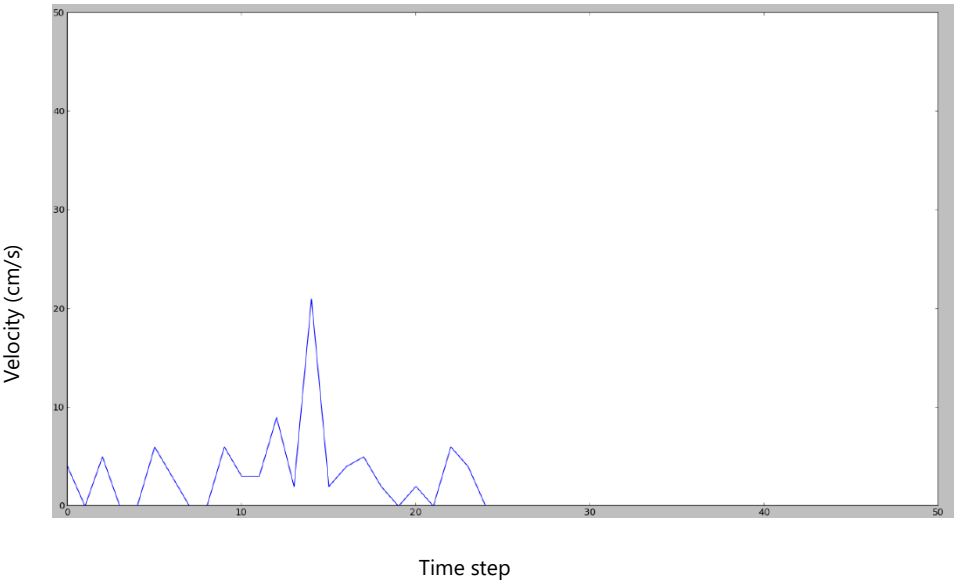
The CPU enters lower power mode for at most 13.1 ms (4 overflows) and sets the velocity to 0 if no measurement is detected. The velocity is sent to the python program that plots a zero reading and saves the value along with its time since the application started running in time_and_velocity.txt. The red LED is then toggled and the MSP430 loops the main application after a delay of 50 clock cycles (20.48 ms).

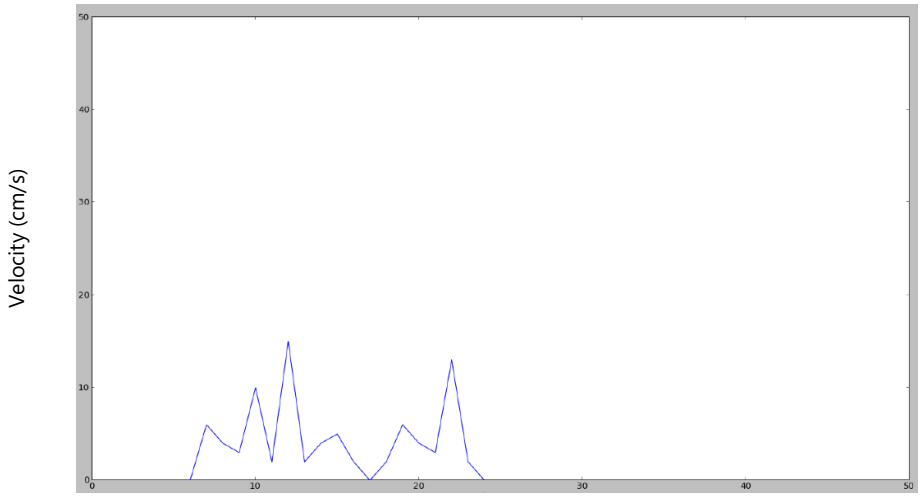
If there is motion, particularly of a hard surface, within a range of 2cm to 20 cm and within a 15 degree measuring angle[5]; the phase detector outputs the difference between the sent and received signal as shown below for a test sample. The channel 1 signal (yellow) is the sent pulse while the channel 2 signal (blue) is the phase detector output.

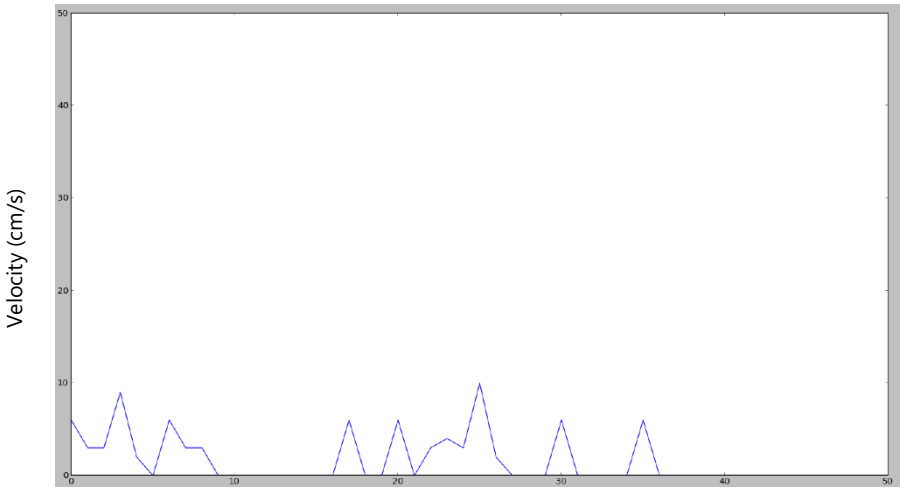


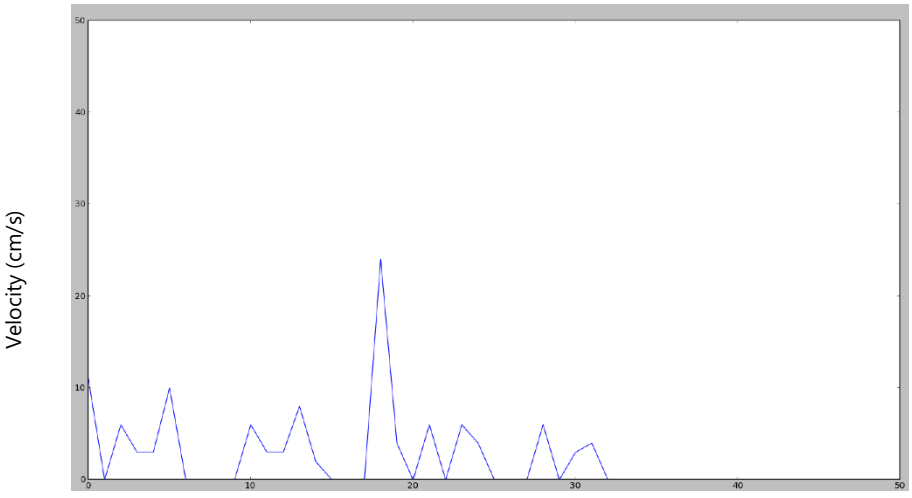
Continuous motion within range results in sample data from time_and_velocity.txt (appendix 3) with graphs below of time intervals with recorded motion.






Velocity (cm/s)
0
10
20
30
40
50
Time step


Velocity (cm/s)
0
10
20
30
40
50
Time step


Velocity (cm/s)
0
10
20
30
40
50
Time step

For distances in range, the velocity readings had large error ranges of up to ± 5 cm/s and a significant noise to data ratio. At distances beyond 20 cm, any moving object within the 15 degree angle of the sensor was not picked up and our plot showed a 0 velocity.

DISCUSSIONS OF THE DEVICE

Writing the interrupts that analyse and collect the data proved to be the most difficult aspect of the project. The high clock frequency necessary to capture the frequency difference meant using dividers on all timers and careful conversions for all appropriate variables including data transmission variables at the right Baud rate (9600/4). Pulse generation, analysis and transmission of data by the MSP430 performed as expected. The ultrasonic distance sensor was the least successful device component. One of the likely causes for its poor performance is the receiver's band pass filter response. Its frequency is centered at 18 kHz while the sent signal is 40 kHz resulting in poor detection sensitivity for the frequency phase difference. Switching the resistor combination in its circuit would improve its frequency response. The filter also offers poor protection against false signals in noisy environments such as the moving hard surfaces that were of interest[8]. More tests should be performed with a different ultrasonic sensor in the same setup.

CONCLUSIONS

While functional, the device is too inconsistent, has a low signal to noise ratio and is too imprecise to be used reliable as a velocity meter even within its limited range. The modified HC-SR04 makes a poor Doppler velocity meter.

REFERENCES

1. Alec Eden *The search for Christian Doppler*,Springer-Verlag, Wien 1992

2. Evans, D. H.; McDicken, W. N. (2000). *Doppler Ultrasound* (2nd ed.). New York: John Wiley and Sons

3. Keel, W. C. (2007). *The Road to Galaxy Formation* (2nd ed.). Springer. pp. 7–8

4. http://uglyduck.ath.cx/HC-SR04E/HC-SR04.svgz

5. http://www.robot-electronics.co.uk/htm/srf04tech.htm

6. Rosen, Joe; Gothard, Lisa Quinn (2009). *Encyclopedia of Physical Science*. p. 155

7. http://cpuville.com/logic_gates.htm

8. http://uglyduck.ath.cx/ep/archive/2014/01/Making_a_better_HC_SR04_Echo_Locator.html

APPENDIX

Appendix 1

```
#include  "msp430.h"

#define     LED1                  BIT0
#define     LED2                  BIT6


#define     BUTTON                BIT3

#define     TXD                   BIT1                      // TXD on P1.1
#define     RXD                   BIT2                      // RXD on P1.2
#define     ONOFF                 BIT4
#define     PreAppMode            0
#define     RunningMode           1
//   Conditions for 9600/4=2400 Baud SW UART, SMCLK = 1MHz

#define     Bitime                833
// duration of 1 bit, measured in timer A clock cycles.
// main clock 16MHZ timer a at 2 MHz
//For 2400 bits per second, we want 833 timer ticks.

unsigned char BitCnt;
unsigned int TXByte;
unsigned char data;
static volatile int Times[4];
volatile int i = 0;

volatile long velocity = 0;
int capture = 0;
int pulsecount =0 ;
volatile int overflowcounter=0;
volatile unsigned int Mode;

void ConfigureTimerUart(void);
void InitializeButton(void);
void PreApplicationMode(void);
void Transmit(void);

void main(void)
{


  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT


  BCSCTL1 = CALBC1_16MHZ;                    // Set DCO to 16MHz
  DCOCTL = CALDCO_16MHZ;
  BCSCTL2 &= ~(DIVS_3);                      // SMCLK = DCO = 16MHz

  InitializeButton();

  // setup port for leds:
  P1DIR |= LED1 + LED2 + ONOFF;
  P1OUT &= ~(LED1 + LED2 + ONOFF);
```

```
  //set ports for TXD
  P1DIR |= TXD;
  P1OUT |= TXD;

  //set ports
  P2DIR |= BIT1+BIT2+BIT3; // PWM , MAX232A ON/OFF , THRESHOLD
  P2SEL |= BIT1;

  Mode = PreAppMode;
  PreApplicationMode();          // Blinks LEDs, waits for button press


  __delay_cycles(1000);                     // Wait for ADC Ref to settle

  __enable_interrupt();                     // Enable interrupts.


  TA1CTL = TASSEL_2 + MC_1 + ID_3; // SMCLK, up mode, with no divider
  TA1CCR0 = 50;
  TA1CCR1 = 25; // Period and Dutycycle for a 25us pulse
  TA1CCTL1 = OUTMOD_7; // CCR1 reset/set


  /* Main Application Loop */
  while(1){

    P2OUT |= BIT2 + BIT3 ;  // Comparator threshold high + turn on sending transmitter
    P1OUT |= ONOFF;   // turn on nand gate

    for(pulsecount=0;pulsecount<400;pulsecount++); // wait to generate pulse train

    P2OUT &= ~(BIT2);
    P1OUT&= ~(ONOFF);
    P2OUT&= ~(BIT3); //threshold low, Nand low, turn off sending transmitter

    capture = 1;           //  Sending  off

    TACTL = TASSEL_2 + MC_2 +ID_3 + TAIE;
    TACCTL1 = CAP + CCIS_0 + CM_3 + CCIE  ;// Enable capture compare for rising and falling
edges and timer_A0 interrupts.

    P1SEL |= RXD; // alternate function for capture compare input pin shared with RXD

    __bis_SR_register(LPM0_bits + GIE); // Enter Low power mode 0

    // set up timer and port for uart.
    capture = 0;
    TACTL &= ~( TAIE);// Disable interrupt.

    ConfigureTimerUart();

    // convert to string  and send to host computer
    TXByte = (unsigned char) (velocity);
    Transmit();

    P1OUT ^= LED1;  // toggle the light every time we make a measurement.
```

```
    // set up timer to wake us in a while:
    __delay_cycles(50);

  }


}

void PreApplicationMode(void)
{
  P1DIR |= LED1 + LED2;
  P1OUT |= LED1;                // To enable the LED toggling effect
  P1OUT &= ~LED2;

  /* these next two lines configure the ACLK signal to come from
     a secondary oscillator source, called VLO */

  BCSCTL1 |= DIVA_1;            // ACLK is half the speed of the source (VLO)
  BCSCTL3 |= LFXT1S_2;          // ACLK = VLO

  /* here we're setting up a timer to fire an interrupt periodically.
     When the timer 1 hits its limit, the interrupt will toggle the lights

     We're using ACLK as the timer source, since it lets us go into LPM3
     (where SMCLK and MCLK are turned off). */

  TACCR0 = 1200;                //  period
  TACTL = TASSEL_1 | MC_1;      // TACLK = ACLK, Up mode.
  TACCTL1 = CCIE + OUTMOD_3;    // TACCTL1 Capture Compare
  TACCR1 = 600;                 // duty cycle
  __bis_SR_register(LPM3_bits + GIE);   // LPM3 with interrupts enabled
  // in LPM3, MCLCK and SMCLK are off, but ACLK is on.
}

void ConfigureTimerUart(){
  TACCTL0 = OUT;                // TXD Idle as Mark
  TACTL = TASSEL_2 + MC_2 + ID_3;// set SMCLK as source, divide by 8, continuous mode
  P1SEL |= (TXD+RXD);
  P1DIR |= TXD;
}

/* using the serial port requires Transmit(),
   the TIMERA0_VECTOR, ConfigureTimerUart()
   and variables Bitcnt, TXbyte, Bitime */

// Function Transmits Character from TXByte
void Transmit()
{

  BitCnt = 0xA;           // Load Bit counter, 8 data + Start/Stop bit
  TXByte |= 0x100;        // Add mark stop bit to TXByte
  TXByte = TXByte << 1;   // Add space start bit

  /*
  // The TI folks originally had the four lines of code below, but why?
  // replace with the single line after the comment ends
```

```
  //  Simulate a timer capture event to obtain the value of TAR into
  //   the TACCR0 register.

  TACCTL0 = CM_1 + CCIS_2  + SCS + CAP + OUTMOD0;  //capture on rising edge,
  // initially set to GND as input
  TACCTL0 |= CCIS_3;  //change input to Vcc, rising the edge,
  // triggering the capture action
  while (!(TACCTL0 & CCIFG));//wait till the interrupt happens.
  TACCR0 += Bitime;                        // Time till first bit
  */

  TACCR0 = TAR+ Bitime;
  TACCTL0 =  CCIS0 + OUTMOD0 + CCIE;   // TXD = mark = idle, enable interrupts
  // OUTMOD0 sets output mode 1: SET which will
  // have the CCR bit (our TX bit) to go high when the timer expires
  while ( TACCTL0 & CCIE );                   // Wait for TX completion
}


// Timer A0 interrupt service routine -
#if defined(__TI_COMPILER_VERSION__)
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
#else
  void __attribute__ ((interrupt(TIMER0_A0_VECTOR))) Timer_A (void)
#endif
{
  TACCR0 += Bitime;                         // Add Offset to TACCR0
  if ( BitCnt == 0){
    P1SEL &= ~(TXD+RXD);
    TACCTL0 &= ~ CCIE ;                   // All bits TXed, disable interrupt
  }
  else{
    // in here we set up what the next bit will be: when the timer expires
    // next time.
    // In TimerConfigUart, we set OUTMOD0 for output mode 1 (set).
    // Adding OUTMOD2 gives output mode 5 (reset).

    // The advantage to doing this is that the bits get set in hardware
    // when the timer expires so the timing is as accurate as possible.
    TACCTL0 |=  OUTMOD2;                  // puts output unit in 'set' mode
    if (TXByte & 0x01)
      TACCTL0 &= ~ OUTMOD2;               // puts output unit in reset mode
    TXByte = TXByte >> 1; // shift down so the next bit is in place.
    BitCnt --;
  }
}




// this gets used in pre-application mode only to toggle the lights:
#if defined(__TI_COMPILER_VERSION__)
#pragma vector=TIMER0_A1_VECTOR
__interrupt void ta1_isr (void)
```

```
#else
  void __attribute__ ((interrupt(TIMER0_A1_VECTOR))) ta1_isr (void)
#endif
{

  if(TACCTL1 & CCIFG){
    if (capture == 0){
      TACCTL1 &= ~CCIFG; // reset the interrupt flag
      if (Mode == PreAppMode){
       P1OUT ^= (LED1 + LED2); // toggle the two lights.
      }
      else{
       TACCTL1 = 0;                // no more interrupts.
       __bic_SR_register_on_exit(LPM3_bits);        // Restart the cpu
      }
    }

    else{
      // if recorded four data points, find delta t and calculate velocity
      if ( i >= 3){

       long delta_total = 0;
       int k;
       long delta_t;
       int m = 0;
       for (k = 0; k + 3<= i; k= k + 2){
         long delta_a  = Times[k + 1]- Times[k];
         long delta_b  = Times[k + 3]- Times[k + 2];
         if (delta_b > delta_a){
           delta_total += delta_b - delta_a ;
           m++;
         }
         else {
           delta_total += delta_a - delta_b ;
           m++;
         }
       }// calculates delta t for an array of any size

       delta_t =  2 *(delta_total)/m;
       velocity = 1000*20000000/(delta_t * 1166); // factor from speed of sound in air (34320
cm/s) and baseline signal of 40kHz to get velocity from doppler effect.
       i = 0;
       m = 0;

       int delay =0 ;
       for(delay=0;delay<20000;delay++);
       TACCTL1 &= ~CCIFG;                  // Reset the input flag

       __bic_SR_register_on_exit( LPM0_bits );           // Clear LMP0 bits so we are ruuning
normal mode

      }
      else{
       TACCTL1 &= ~CCIFG;  // Reset the input flag
       Times[i] = TACCR1;  // Record time measurement
       i++;
      }
```

```
    }


  }
  else{
    TACTL &= ~TAIFG;
    if(overflowcounter<4){
      overflowcounter++;

    }else{
      overflowcounter=0;
      velocity = 0;
      TACTL &= ~TAIE;
      __bic_SR_register_on_exit( LPM0_bits ); // If no velocity reading is available, exit low
power mode after a count of 4 timer A overflows.
    }
  }
}

void InitializeButton(void)                // Configure Push Button
{
  P1DIR &= ~BUTTON;
  P1OUT |= BUTTON;
  P1REN |= BUTTON;
  P1IES |= BUTTON;
  P1IFG &= ~BUTTON;
  P1IE |= BUTTON;
}

/* ************************************************************
 * Port Interrupt for Button Press
 * 1. During standby mode: to enter application mode
 *
 * ************************************************************ */


#if defined(__TI_COMPILER_VERSION__)
#pragma vector=PORT1_VECTOR
__interrupt void port1_isr(void)
#else
  void __attribute__ ((interrupt(PORT1_VECTOR))) port1_isr (void)
#endif

{

  /* this disables port1 interrupts for a little while so that
  we don't try to respond to two consecutive button pushes right together.
  The watchdog timer interrupt will re-enable port1 interrupts

  This whole watchdog thing is completely unnecessary here, but its useful
  to see how it is done.
*/
  P1IFG = 0;  // clear out interrupt flag
  P1IE &= ~BUTTON;         // Disable port 1 interrupts
  WDTCTL = WDT_ADLY_250;   // set up watchdog timer duration
  IFG1 &= ~WDTIFG;         // clear interrupt flag
  IE1 |= WDTIE;            // enable watchdog interrupts
```

```
  TACCTL1 = 0;             // turn off timer 1 interrupts
  P1OUT &= ~(LED1+LED2);   // turn off the leds
  Mode = RunningMode;
  __bic_SR_register_on_exit(LPM3_bits); // take us out of low power mode

}

// WDT Interrupt Service Routine used to de-bounce button press

#if defined(__TI_COMPILER_VERSION__)
#pragma vector=WDT_VECTOR
__interrupt void wdt_isr(void)
#else
  void __attribute__ ((interrupt(WDT_VECTOR))) wdt_isr (void)
#endif

{
    IE1 &= ~WDTIE;                  /* disable watchdog interrupt */
    IFG1 &= ~WDTIFG;                /* clear interrupt flag */
    WDTCTL = WDTPW + WDTHOLD;       /* put WDT back in hold state */
    P1IE |= BUTTON;             /* Debouncing complete - reenable port 1 interrupts*/
}
```

Appendix 2

```
#!/usr/bin/python2.7
import serial # for serial port
import numpy as np # for arrays, numerical processing
from time import sleep,time
import gtk #the gui toolkit we'll use:
# graph plotting library:
from matplotlib.figure import Figure
from matplotlib.backends.backend_gtkagg import FigureCanvasGTKAgg as FigureCanvas

#needs: python2, pyserial, numpy, matplotlib, pygtk
#0) flash the serial temperature measurement program into the msp430
#1) start this program
#2) press the button the Launchpad to enter 'applcation mode'
#3) warm the chip (eg with a light bulb or your fingers)
#4) when you've seen enough, press the reset button on the launchpad
#5) exit the program by pressing 'q' or clicking on the x

#define the serial port. Pick one:
#port = "/dev/ttyACM0"  #for Linux
port = "COM5" #For Windows?
#port = "/dev/tty.uart-XXXX" #For Mac?

#function that gets called when a key is pressed:
def press(event):
    print('press', event.key)
    if event.key == 'q':
        print ('got q!')
        quit_app(None)
    return True
```

```
def quit_app(event):
    outFile.close()
    ser.close()
    quit()

#start our program proper:
#open the serial port
try:
    # It seems that sometimes the port doesn't work unless
    # you open it first with one speed, then change it to the correct value
    ser = serial.Serial(port,9600,timeout = 0.050)
    ser.baudrate=2400
# with timeout=0, read returns immediately, even if no data
except:
    print ("Opening serial port",port,"failed")
    print ("Edit program to point to the correct port.")
    print ("Hit enter to exit")
    raw_input()
    quit()

#create a window to put the plot in
win = gtk.Window()
#connect the destroy signal (clicking the x in the corner)
win.connect("destroy", quit_app)
win.set_default_size(400,300)

yvals = np.zeros(50) #array to hold last 50 measurements
times=np.arange(0,50,1.0) # 50 from 0 to 49.

#create a plot:
fig = Figure()
ax = fig.add_subplot(111,xlabel='Time Step',ylabel='Velocity')
ax.set_ylim(0,50) # set limits of y axis.

canvas = FigureCanvas(fig) #put the plot onto a canvas
win.add(canvas) #put the canvas in the window

# define a callback for when a key is pressed
fig.canvas.mpl_connect('key_press_event',press)

#show the window
win.show_all()
win.set_title("ready to receive data");

line, = ax.plot(times,yvals)
#open a data file for the output
outFile = open("time_and_velocity.txt","w")
start_time = time()
ser.flushInput()

while(1): #loop forever
    data = ser.read(1) # look for a character from serial port - will wait for up to 50ms
(specified above in timeout)
    if len(data) > 0: #was there a byte to read?
        yvals = np.roll(yvals,-1) # shift the values in the array
        yvals[49] = ord(data) # take the value of the byte
        outFile.write(str(time()-start_time)+" "+str(yvals[49])+"\n") #write to file
```

```
        line.set_ydata(yvals) # draw the line
        fig.canvas.draw() # update the canvas
        win.set_title(str(yvals[49]))
    while gtk.events_pending():     #makes sure the GUI updates
        gtk.main_iteration()
```

Appendix 3

Time - Velocity

6.80800008774 0.0	12.5959999561 0.0	18.3870000839 0.0	22.8980000019 0.0	30.986000061 0.0	37.4240000248
6.97399997711 0.0	12.7539999485 0.0	18.5539999008 0.0	22.9360001087 2.0	31.1459999084 0.0	15.0
7.1400001049 0.0	12.7850000858 6.0	18.7149999142 0.0	22.9670000076 0.0	31.3190000057 0.0	37.4620001316 2.0
7.2990000248 0.0	12.9300000668 0.0	18.8770000935 0.0	22.9990000725 6.0	31.4700000286 0.0	37.4930000305 4.0
7.46000003815 0.0	13.0840001106 0.0	19.0460000038 0.0	23.0360000134 4.0	31.6370000839 0.0	37.5250000954 5.0
7.63000011444 0.0	13.253000021 0.0	19.2030000687 0.0	23.0989999771 0.0	31.8029999733 0.0	37.5620000362 2.0
7.79999995232 0.0	13.4160001278 0.0	19.370000124 0.0	23.2539999485 0.0	31.9790000916 0.0	37.5940001011 0.0
7.96200013161 0.0	13.5850000381 0.0	19.5399999619 0.0	23.4200000763 0.0	32.1300001144 0.0	37.625 2.0
8.11899995804 0.0	13.7550001144 0.0	19.7090001106 0.0	23.5889999866 0.0	32.3029999733 0.0	37.6630001068 6.0
8.28600001335 0.0	13.9170000553 0.0	19.8580000401 0.0	23.757999897 0.0	32.4630000591 0.0	37.6940000057 4.0
8.45499992371 0.0	14.0699999332 0.0	19.9100000858 6.0	23.9089999199 0.0	32.635999918 0.0	37.7249999046 3.0
8.61599993706 0.0	14.2479999065 0.0	20.0309998989 0.0	24.0799999237 0.0	32.7990000248 0.0	37.7630000114
8.78299999237 0.0	14.4059998989 0.0	20.1949999332 0.0	24.2520000935 0.0	32.9570000172 0.0	13.0
8.9470000267 0.0	14.5769999027 0.0	20.364000082 0.0	24.4160001278 0.0	33.1259999275 0.0	37.7939999104 2.0
9.11599993706 0.0	14.7400000095 0.0	20.3949999809 6.0	24.5710000992 0.0	33.2939999104 0.0	37.8259999752 0.0
9.27900004387 0.0	14.8940000534 0.0	20.4170000553 4.0	24.7369999886 0.0	33.4460000992 0.0	37.8629999161 0.0
9.43600010872 0.0	15.0659999847 0.0	20.4479999542 3.0	24.9040000439 0.0	33.6159999371 0.0	37.9960000515 0.0
9.60299992561 0.0	15.2290000916 0.0	20.4790000916	25.0599999428 0.0	33.7780001163 0.0	38.1530001163 0.0
9.77099990845 0.0	15.3980000019 0.0	15.0	25.2239999771 0.0	33.9470000267 0.0	38.3199999332 0.0
9.9319999218 0.0	15.5480000973 0.0	20.5169999599 2.0	25.3940000534 0.0	34.1159999371 0.0	38.489000082 0.0
10.0950000286 0.0	15.7179999352 0.0	20.5640001297 0.0	25.5669999123 0.0	34.2799999714 0.0	38.6579999924 0.0
10.2660000324 0.0	15.8910000324 0.0	20.5950000286 6.0	25.7179999352 0.0	34.4319999218 0.0	38.8129999638 0.0
10.2809998989 6.0	15.9219999313 7.0	20.7190001011 0.0	25.888999939 0.0	34.4639999866 6.0	38.986000061 0.0
10.3120000362 4.0	15.9539999962 4.0	20.8849999905 0.0	26.0520000458 0.0	34.5020000935 4.0	39.1380000114 0.0
10.3489999771 3.0	15.9749999046 3.0	20.9230000973 6.0	26.2100000381 0.0	34.6059999466 0.0	39.3029999733 0.0
10.3959999084	16.0220000744	21.0550000668 0.0	26.385999918 0.0	34.6370000839 6.0	39.4800000191 0.0
11.0	12.0	21.0859999657 6.0	26.5410001278 0.0	34.7720000744 0.0	39.6300001144 0.0
10.4279999733 2.0	16.0529999733 2.0	21.2230000496 0.0	26.7149999142 0.0	34.7880001068 6.0	39.8029999733 0.0
10.4490001202	16.0910000801	21.3789999485 0.0	26.878000021 0.0	34.9300000668 0.0	39.9700000286 0.0
15.0	12.0	21.4100000858 1.0	27.0320000648 0.0	34.9609999657 6.0	40.1229999065 0.0
10.4800000191 2.0	16.121999979 3.0	21.5480000973 0.0	27.2030000687 0.0	35.0940001011 0.0	40.2999999523 0.0
10.510999918 0.0	16.1530001163 0.0	21.7109999657 0.0	27.3629999161 0.0	35.131000042 6.0	40.4509999752 0.0
10.5490000248 6.0	16.1909999847 6.0	21.7330000401 6.0	27.5320000648 0.0	35.2630000114 0.0	40.628000021 0.0
10.5799999237 4.0	16.2220001221 3.0	21.763999939 4.0	27.7009999752 0.0	35.2939999104 6.0	40.7939999104 0.0
10.6119999886 3.0	16.253000021 3.0	21.8800001144 0.0	27.8589999676 0.0	35.3159999847 3.0	40.9479999542 0.0
10.6489999294 0.0	16.2910001278 0.0	21.9110000134 5.0	28.0220000744 0.0	35.4319999218 0.0	41.1210000515 0.0
10.6809999943 6.0	16.4100000858 0.0	22.0329999924 0.0	28.1840000153 0.0	35.5940001011 0.0	41.2880001068 0.0
10.7799999714 0.0	16.5840001106 0.0	22.1949999332 0.0	28.3559999466 0.0	35.7479999065 0.0	41.4490001202 0.0
10.8120000362 6.0	16.5999999046 6.0	22.2330000401 6.0	28.5180001259 0.0	35.9210000038 0.0	41.6150000095 0.0
10.8489999771 4.0	16.742000103 0.0	22.263999939 3.0	28.6840000153 0.0	36.0880000591 0.0	41.7690000534 0.0
10.9530000687 0.0	16.7780001163 6.0	22.364000082 0.0	28.8529999256 0.0	36.2490000725 0.0	41.9400000572 0.0
11.1150000095 0.0	16.9110000134 0.0	22.5350000858 0.0	29.0039999485 0.0	36.4079999924 0.0	42.1029999256 0.0
11.2829999924 0.0	17.0780000687 0.0	22.5659999847 6.0	29.1800000668 0.0	36.5789999962 0.0	42.2720000744 0.0
11.4409999847 0.0	17.2300000191 0.0	22.5980000496 3.0	29.3370001316 0.0	36.7369999886 0.0	42.4390001297 0.0
11.6070001125 0.0	17.3989999294 0.0	22.6349999905 3.0	29.503000021 0.0	36.9040000439 0.0	42.5920000076 0.0
11.6449999809 6.0	17.5640001297 0.0	22.6670000553 9.0	29.6719999313 0.0	37.0729999542 0.0	42.760999918 0.0
11.6760001183 4.0	17.5999999046 6.0	22.6979999542 2.0	29.8420000076 0.0	37.2409999371 0.0	42.9270000458 0.0
11.7769999504 0.0	17.631000042 4.0	22.7350001335	29.9930000305 0.0	37.2620000839 6.0	43.0940001011 0.0
11.9460000992 0.0	17.6630001068 3.0	21.0	30.1700000763 0.0	37.2929999828 4.0	43.25 0.0
12.0959999561 0.0	17.7320001125 0.0	22.7669999599 2.0	30.3199999332 0.0	37.3240001202 3.0	43.4119999409 0.0
12.1270000935 3.0	17.888999939 0.0	22.7980000973 4.0	30.489000082 0.0	37.3619999886	43.5810000896 0.0
12.2650001049 0.0	18.0640001297 0.0	22.8359999657 5.0	30.6619999409 0.0	10.0	43.7539999485 0.0
12.4279999733 0.0	18.2179999352 0.0	22.867000103 2.0	30.8129999638 0.0	37.3930001259 2.0	43.9059998989 0.0

44.0740001202 0.0	54.6050000191 0.0	62.8510000706 0.0	68.3519999981 0.0	78.878000021 0.0	89.3959999084 0.0
44.2409999371 0.0	54.7569999695 0.0	63.0120000839 0.0	68.5130000114 0.0	79.0320000648 0.0	89.5529999733 0.0
44.4119999409 0.0	54.9340000153 0.0	63.1789999008 0.0	68.6789999008 0.0	79.2049999237 0.0	89.7200000286 0.0
44.5669999123 0.0	55.0959999561 0.0	63.3499999046 0.0	68.8480000496 0.0	79.3710000515 0.0	89.8910000324 0.0
44.7290000916 0.0	55.25 0.0	63.5190000534 0.0	69.0150001049 0.0	79.5269999504 0.0	90.0520000458 0.0
44.8970000744 0.0	55.4219999313 0.0	63.6700000763 0.0	69.1789999008 0.0	79.6860001087 0.0	90.2179999352 0.0
45.0659999847 0.0	55.5889999866 0.0	63.8470001221 0.0	69.3359999657 0.0	79.8550000191 0.0	90.3759999275 0.0
45.2290000916 0.0	55.7490000725 0.0	64.010999918 0.0	69.5069999695 0.0	80.0290000439 0.0	90.4070000648 6.0
45.3980000019 0.0	55.9149999619 0.0	64.1649999619 0.0	69.6719999313 0.0	80.1800000668 0.0	90.5420000553 0.0
45.5629999638 0.0	56.0820000172 0.0	64.3399999142 0.0	69.8289999962 0.0	80.3529999256 0.0	90.7130000591 0.0
45.7130000591 0.0	56.2400000095 0.0	64.5050001144 0.0	69.9869999886 0.0	80.5199999809 0.0	90.870000124 0.0
45.8819999695 0.0	56.4030001163 0.0	64.6559998989 0.0	70.1649999619 0.0	80.6749999523 0.0	90.9019999504 7.0
46.0460000038 0.0	56.5720000267 0.0	64.8210000992 0.0	70.3159999847 0.0	80.8510000706 0.0	91.0369999409 0.0
46.2149999142 0.0	56.7430000305 0.0	64.9900000095 0.0	70.489000082 0.0	81.003000021 0.0	91.0740001202 6.0
46.3849999905 0.0	56.8980000019 0.0	65.1630001068 0.0	70.6500000954 0.0	81.1800000668 0.0	91.1059999466 4.0
46.5480000973 0.0	57.0659999847 0.0	65.1789999008 6.0	70.8169999123 0.0	81.3420000076 0.0	91.1370000839 3.0
46.7009999752 0.0	57.2290000916 0.0	65.2260000706 3.0	70.9869999886 0.0	81.4960000515 0.0	91.2100000381 0.0
46.870000124 0.0	57.3980000019 0.0	65.260999918 3.0	71.1519999504 0.0	81.6679999828 0.0	91.2249999046 6.0
47.0339999199 0.0	57.5520000458 0.0	65.2950000763 9.0	71.3039999008 0.0	81.8350000381 0.0	91.2739999294 3.0
47.2030000687 0.0	57.7149999142 0.0	65.3259999752 2.0	71.4730000496 0.0	81.9960000515 0.0	91.375 0.0
47.3710000515 0.0	57.8849999905 0.0	65.3629999161	71.638999939 0.0	82.1610000134 0.0	91.3910000324 6.0
47.5350000858 0.0	58.0529999733 0.0	18.0	71.8090000153 0.0	82.3159999847 0.0	91.4379999638 3.0
47.6890001297 0.0	58.2130000591 0.0	65.3949999809 0.0	71.9620001316 0.0	82.4869999886 0.0	91.4709999561 3.0
47.8529999256 0.0	58.3849999905 0.0	65.5130000114 0.0	72.128000021 0.0	82.6530001163 0.0	91.4909999371
48.0220000744 0.0	58.5510001183 0.0	65.6860001087 0.0	72.2969999313 0.0	82.8229999542 0.0	10.0
48.1919999123 0.0	58.7039999962 0.0	65.7170000076 3.0	72.4670000076 0.0	82.9879999161 0.0	91.5220000744 2.0
48.3550000191 0.0	58.871999979 0.0	65.7479999065 4.0	72.618999958 0.0	83.1430001259 0.0	91.5729999542 0.0
48.507999897 0.0	59.0369999409 0.0	65.7860000134 3.0	72.7960000038 0.0	83.3029999733 0.0	91.5910000801 6.0
48.6830000877 0.0	59.2060000896 0.0	65.8169999123	72.9460000992 0.0	83.4700000286 0.0	91.6229999065 3.0
48.8450000286 0.0	59.3600001335 0.0	11.0	73.114000082 0.0	83.638999939 0.0	91.6740000248 3.0
49.0150001049 0.0	59.5260000229 0.0	65.8489999771 0.0	73.2869999409 0.0	83.8110001087 0.0	91.7070000172 9.0
49.1800000668 0.0	59.6960000992 0.0	65.885999918 6.0	73.4539999962 0.0	83.9620001316 0.0	91.739000082 2.0
49.3350000381 0.0	59.8659999371 0.0	65.9179999828 3.0	73.611000061 0.0	84.135999918 0.0	91.7739999294 0.0
49.5009999275 0.0	60.0180001259 0.0	65.9490001202 3.0	73.7780001163 0.0	84.3029999733 0.0	91.7920000553 6.0
49.6619999409 0.0	60.1909999847 0.0	65.9869999886	73.9319999218 0.0	84.4679999352 0.0	91.8389999866 3.0
49.8310000896 0.0	60.3570001125 0.0	10.0	74.1059999466 0.0	84.632999897 0.0	91.8759999275 3.0
50.0020000935 0.0	60.510999918 0.0	66.0180001259 0.0	74.2730000019 0.0	84.7960000038 0.0	91.9230000973 0.0
50.1530001163 0.0	60.5420000553 6.0	66.1710000038 0.0	74.4340000153 0.0	84.9530000687 0.0	92.0999999046 0.0
50.3159999847 0.0	60.6809999943 0.0	66.3380000591 0.0	74.6010000706 0.0	85.117000103 0.0	92.2509999275 0.0
50.493999958 0.0	60.8459999561 0.0	66.5069999695 0.0	74.7550001144 0.0	85.2869999409 0.0	92.4240000248 0.0
50.6489999294 0.0	61.0160000324 0.0	66.5380001068 6.0	74.9330000877 0.0	85.4530000687 0.0	92.5940001011 0.0
50.8229999542 0.0	61.1819999218 0.0	66.5759999752 3.0	75.0950000286 0.0	85.6040000916 0.0	92.746999979 0.0
50.9739999771 0.0	61.3359999657 0.0	66.6070001125 3.0	75.25 0.0	85.7809998989 0.0	92.9200000763 0.0
51.1500000954 0.0	61.5099999905 0.0	66.6380000114 8.0	75.4119999409 0.0	85.9430000782 0.0	93.0859999657 0.0
51.3159999847 0.0	61.5260000229 6.0	66.6760001183 2.0	75.5850000381 0.0	86.0970001221 0.0	93.1089999676 6.0
51.4690001011 0.0	61.5729999542 4.0	66.7070000172 0.0	75.7460000515 0.0	86.2699999809 0.0	93.239000082 0.0
51.6419999599 0.0	61.5940001011 3.0	66.8759999275 0.0	75.9130001068 0.0	86.4370000362 0.0	93.4160001278 0.0
51.8090000153 0.0	61.6410000324	67.0299999714 0.0	76.0829999447 0.0	86.5940001011 0.0	93.4319999218 6.0
51.9700000286 0.0	11.0	67.0610001087	76.2339999676 0.0	86.760999918 0.0	93.5780000687 0.0
52.135999918 0.0	61.6719999313 2.0	24.0	76.4049999714 0.0	86.9300000668 0.0	93.6100001335 3.0
52.2899999619 0.0	61.7100000381	67.0989999771 4.0	76.5659999847 0.0	87.0870001316 0.0	93.632999897 4.0
52.4679999352 0.0	12.0	67.2009999752 0.0	76.7290000916 0.0	87.25 0.0	93.6640000343 3.0
52.620000124 0.0	61.7409999371 0.0	67.2320001125 6.0	76.8989999294 0.0	87.4230000973 0.0	93.6949999332
52.7820000648 0.0	61.7720000744 6.0	67.3659999371 0.0	77.0710000992 0.0	87.5920000076 0.0	10.0
52.9609999657 0.0	61.871999979 0.0	67.3880000114 6.0	77.2350001335 0.0	87.746999979 0.0	93.7330000401 2.0
53.1150000095 0.0	62.0260000229 0.0	67.4200000763 4.0	77.3910000324 0.0	87.9200000763 0.0	93.763999939 0.0
53.2890000343 0.0	62.0569999218 6.0	67.5199999809 0.0	77.5590000153 0.0	88.0710000992 0.0	93.9360001087 0.0
53.4400000572 0.0	62.1990001202 0.0	67.6970000267 0.0	77.7279999256 0.0	88.2400000095 0.0	94.1040000916 0.0
53.614000082 0.0	62.2300000191 6.0	67.8619999886 0.0	77.882999897 0.0	88.4140000343 0.0	94.125 6.0
53.7780001163 0.0	62.2620000839 4.0	67.878000021 6.0	78.0460000038 0.0	88.5629999638 0.0	94.2599999905 0.0
53.9460000992 0.0	62.3619999886 0.0	68.0150001049 0.0	78.2149999142 0.0	88.7320001125 0.0	94.4360001087 0.0
54.0999999046 0.0	62.3970000744 3.0	68.0469999313 3.0	78.385999918 0.0	88.8989999294 0.0	94.5870001316 0.0
54.2620000839 0.0	62.5309998989 0.0	68.0780000687 4.0	78.5369999409 0.0	89.0680000782 0.0	94.763999939 0.0
54.4319999218 0.0	62.6960000992 0.0	68.1789999008 0.0	78.7079999447 0.0	89.2290000916 0.0	94.7950000763 6.0

94.9279999733 0.0
95.0810000896 0.0
95.2539999485 0.0
95.4200000763 0.0
95.5810000896 0.0
95.7400000095 0.0
95.9130001068 0.0
96.0710000992 0.0
96.2369999886 0.0
96.4070000648 0.0
96.5740001202 0.0
96.7300000191 0.0
96.9030001163 0.0
97.0539999008 0.0
97.2230000496 0.0
97.3919999599 0.0
97.5599999428 0.0
97.7249999046 0.0
97.8770000935 0.0
98.0450000763 0.0
98.2079999447 0.0
98.378000021 0.0
98.5320000648 0.0
98.7109999657 0.0
98.864000082 0.0
99.0339999199 0.0
99.1960000992 0.0
99.3650000095 0.0
99.5339999199 0.0
99.6830000877 0.0
99.8589999676 0.0
100.01699996 0.0
100.186000109 0.0
100.342000008 0.0
100.519999981 0.0
100.674000025 0.0
100.842999935 0.0
101.006000042 0.0