Rotating Ultrasonic Distance Sensor
Ben Weinberg & Zhengyuan Cai
5/3/19 ECE:3360 Embedded Systems

1) Introduction

The device we designed and built was inspired by the characteristics of the ultrasonic waves. When the waves make contact with an object, they bounce off of the object and back to the receiver. Using the speed of sound, you can calculate the distance from the object.The device we designed is very similar to something like a radar. By rotating a singular ultrasonic sensor on a servo motor, it is able to tell when an object is in front of the device by calculating the distance to the object keeping the device informed of the surroundings.

2) Implementation

2a) Overview

Our design for this device consisted of the microcontroller (ATmega88PA), the Ultrasonic Distance Sensor (HC-SR04), and the Servo Motor (KY66). When the Ultrasonic Distance Sensor receives a 10us trigger pulse, it will output an echo pulse back to the microcontroller. Measuring the length of this pulse provides the microcontroller with the ability to calculate the distance. By mounting this sensor to a servo, we are able to take the distance at several different points in front of the device. The Servo Motor takes PWM input to determine the position it will rotate to. Out device will rotate 30° at a time from starting at 0° and stepping up to 180°. These measurements are then sent back to the microcontroller and output to the serial (RS232) communication for display on the computer.

2b) Device details

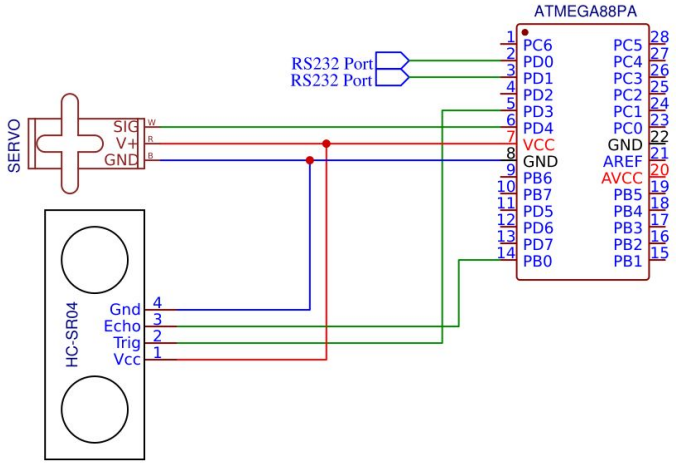


Figure 1. Schematic of device

For this device, we used a HC-SR04 ultrasonic distance sensor, an ATMega88PA microcontroller, and a KY66 servo. To control the servo it needs about a 50 KHz pulse controlled with PWM going into the servo. When the “high” part of the signal is active for 1.5 ms the servo turns to the 0 degree position. When the signal is “high” for 2.5 ms, the servo spins to the 180 degrees position. By using different timings, we rotated the servo in 30 degree

increments to take measurements from the ultrasonic sensor at different angles. The ultrasonic sensor takes a 10us trigger pulse from the microcontroller to start the distance finding sequence. Below is the timing diagram for the device.

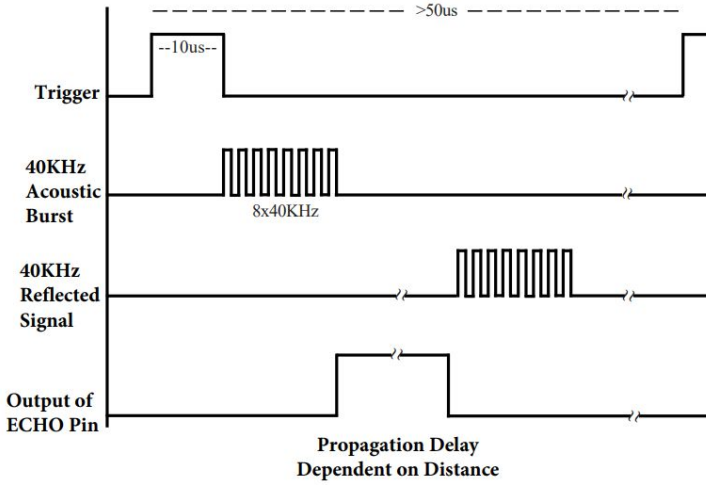


Figure 2. timing diagram for Ultrasonic distance sensor

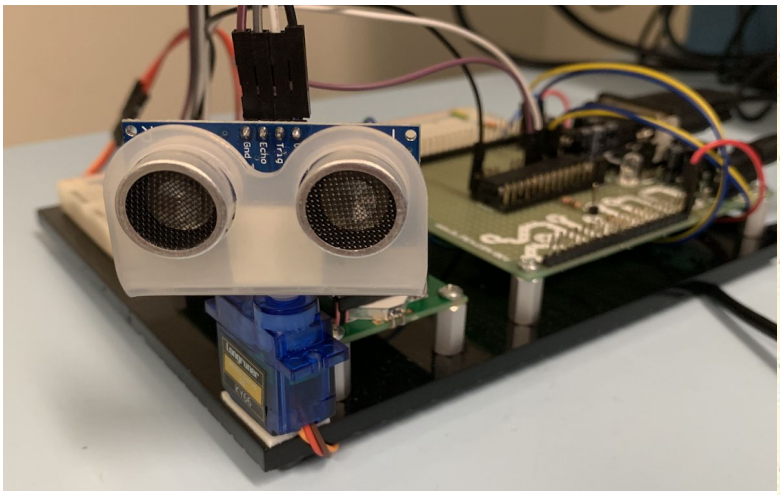
After the trigger pulse is sent to the sensor, several ultrasonic waves are sent out, and while the sensor waits for the return of the waves, it pulls the echo line high. By timing how long this pulse is high, you know how long the wave was in the air for. Using the time the wave was in the air and the speed of sound, you can calculate the distance to the object. We calculate the length of the pulse by utilizing the Input Capture mechanism of Timer1 on the microcontroller. This allows us to get an accurate count of how long the pulse was high for. The distance and current angle is then output to RS232 communication for the serial monitor of a computer to pick up.

3) Experimental Methods

To test out our device, we set up objects at set distances and angles from the device, we then verified the the actual results matched the expected results. Using this method we figured out that our device was accurate up to around 1m. This range can be extended given more time (up to 4m). Using this testing method, we also found that it was hard to interpret the results when they are in graph form and it was much easier to quickly identify the results when they are output in a text format.

4) Results

Shown below (left) is the the ultrasonic distance sensor mounted on a servo motor. The collected data is then sent back from the ultrasonic sensor and is then shown in the serial monitor (below, right).



```
Angle: 0 Degrees
Distance: 14.594722 cm

Angle: 30 Degrees
Distance: 47.244194 cm

Angle: 60 Degrees
Distance: 116.873540 cm

Angle: 90 Degrees
Distance: 104.568350 cm

Angle: 120 Degrees
Distance: 92.233154 cm

Angle: 150 Degrees
Distance: 62.936523 cm

Angle: 180 Degrees
Distance: 62.216221 cm
```

I learned several lessons in constructing our device, I learned more about using the timers built into the ATMega88 and how they work. Also, I learned how to use both PWM and input capture on a timer. We also learned the importance of looking at the datasheets for the timings of each device, and taking time to figure out how to use two different timers for different components.

5) Discussion of Results

Overall the project was very successful. The ultrasonic distance sensor is accurate in getting distances, and the servo appropriately spins to the correct position for measurements. The communication using RS232 between the device and the computer works perfectly and displays the distance measurements with the angle in a readable way. In the future, if I wanted to improve on this design I could add more user interface options, buttons for start/stop, a dial to set a measurement, option to output to LCD rather than RS232. Currently, as stated before, the device only functions well up to around 1m, this can be improved upon in the future by using more memory for the sensor to accommodate larger values. Also, commands could be added using RS232 for specific control of the device.

6) Conclusion

The design is successful and functions properly. The ability for us to learn how more external devices function without much guidance was very enjoyable. In the future, the motor could be swapped out for one that can rotate a full 360° to give distances all the way around an

object rather than just in front of it. This device could be attached to some type of device that will be moving to give it an idea of the surroundings so it does not bump into anything.

7) Acknowledgements

Embedded Systems Lectures on Timings, Interrupts, C Programing, and Serial Communication by Reinhard R. Beichel

8) References/Datasheets

HC-SR04 (ultrasonic distance sensor):
https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

KY66 (servo motor (same as SG90))
http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf

9) Appendix - Source Code

```
#define F_CPU 8000000                    //telling controller crystal frequency attached

#include <avr/io.h>                      //header to enable data flow control over pins
#include <stdlib.h>
#include <util/delay.h>                  //header to enable delay function in program
#include <avr/interrupt.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <avr/pgmspace.h>     //  Routine for FLASH (program memory)

#define BAUD_RATE 9600        // Baud rate. The usart_int routine

// Variables and #define for the RX ring buffer.
#define RX_BUFFER_SIZE 64
unsigned char rx_buffer[RX_BUFFER_SIZE];
volatile unsigned char rx_buffer_head;
volatile unsigned char rx_buffer_tail;

// Function prototypes.
unsigned char uart_buffer_empty(void);
void usart_prints(const char *ptr);
void usart_printf(const char *ptr);
void usart_init(void);
```

```
void usart_putc(const char c);
unsigned char usart_getc(void);
void getDistAndPrint(const int angle);

#define Trigger_pin		PIND7 // trigger pin for us sensor
#define SERVO_PIN		PIND4 // servo "PWM" output
#define SERVO_SET_LOOPS 20		// number of times to send "PWM" pulse

int TimerOverflow = 0;

ISR(TIMER1_OVF_vect)
{
	TimerOverflow++;			// increment timer overflow count for long pulses
}

int main(void)
{

	usart_init();                                                   // initialize
serial to computer


	DDRD |= (1<<Trigger_pin) | (1<<SERVO_PIN);		// setup the servo pin and
trigger pin as an output
	PORTD = 0xFF;                                                   // turn
on pull-up

	sei();                                                          //
enable global interrupt
	TIMSK1 = (1 << TOIE1);                                          // timer1
overflow interrupts
	TCCR1A = 0;                                                     // set
all bits to 0, normal operation

	while(1)
	{

		// set servo to 0 degrees
		for(int i = 0; i < SERVO_SET_LOOPS; i++){
			PORTD |= (1<<SERVO_PIN);
			_delay_us(570);
			PORTD &=~(1<<SERVO_PIN);
			_delay_us(20000);
```

```
}

getDistAndPrint(0);

// set servo to 30 degrees
for(int i = 0; i < SERVO_SET_LOOPS; i++){
        PORTD |= (1<<SERVO_PIN);
        _delay_us(833);
        PORTD &=~(1<<SERVO_PIN);
        _delay_us(20000);
}

getDistAndPrint(30);

// set servo to 60 degrees
for(int i = 0; i < SERVO_SET_LOOPS; i++){
        PORTD |= (1<<SERVO_PIN);
        _delay_us(1166);
        PORTD &=~(1<<SERVO_PIN);
        _delay_us(20000);
}

getDistAndPrint(60);

// set servo to 90 degrees
for(int i = 0; i < SERVO_SET_LOOPS; i++){
        PORTD |= (1<<SERVO_PIN);
        _delay_us(1500);
        PORTD &=~(1<<SERVO_PIN);
        _delay_us(20000);
}

getDistAndPrint(90);

// set servo to 120 degrees
for(int i = 0; i < SERVO_SET_LOOPS; i++){
        PORTD |= (1<<SERVO_PIN);
        _delay_us(1833);
        PORTD &=~(1<<SERVO_PIN);
        _delay_us(20000);
}

getDistAndPrint(120);
```

```
                // set servo to 150 degrees
                for(int i = 0; i < SERVO_SET_LOOPS; i++){
                        PORTD |= (1<<SERVO_PIN);
                        _delay_us(2166);
                        PORTD &=~(1<<SERVO_PIN);
                        _delay_us(20000);
                }

                getDistAndPrint(150);

                //set servo to 180 degrees
                for(int i = 0; i < SERVO_SET_LOOPS; i++){
                        PORTD |= (1<<SERVO_PIN);
                        _delay_us(2500);
                        PORTD &=~(1<<SERVO_PIN);
                        _delay_us(19000);
                }

                getDistAndPrint(180);

        }
}


void getDistAndPrint(const int angle){
        _delay_ms(50); // insure the servo is set

        char string[50];
        long count;
        double distance;

        // send 10us trigger to the ultrasonic sensor
        PORTD |= (1 << Trigger_pin);
        _delay_us(10);
        PORTD &= (~(1 << Trigger_pin));

        // setup timer
        TCNT1 = 0;
        TCCR1B = 0x41;
        TIFR1 |= (1<<ICF1);
        TIFR1 |= (1<<TOV1);
```

```
	// Calculate width of Echo by Input Capture (ICP)
	while ((TIFR1 & (1 << ICF1)) == 0);                                    // Wait for rising edge
(echo to go high)
	TCNT1 = 0;
// Clear timer
	TCCR1B = 0x01;
// Capture on falling edge, No prescaler
	TIFR1 |= (1<<ICF1);                                                    // Clear
ICP flag
	TIFR1 |= (1<<TOV1);                                                    // Clear
Timer Overflow flag
	TimerOverflow = 0;                                                     // Clear
Timer overflow count

	while ((TIFR1 & (1 << ICF1)) == 0);                                    // Wait for echo to go
low
	count = ICR1 + (65535 * TimerOverflow);                                // Take total number
of counts
	distance = (double)count / 466.47;                                     // With 8MHz freq and
speed of sound as 343m/s, calculate distance

	// print the distance to serial
	sprintf(string, "Angle: %i Degrees \r\nDistance: %lf cm \r\n\r\n", angle, distance);
	usart_prints(string);
}


void usart_init(void)
{
	// Configures the USART for serial 8N1 with
	// the Baud rate controlled by a #define.

	unsigned short s;

	// Set Baud rate, controlled with #define above.

	s = (double)F_CPU / (BAUD_RATE*16.0) - 1.0;
	UBRR0H = (s & 0xFF00);
	UBRR0L = (s & 0x00FF);

	// Receive complete interrupt enable: RXCIE0
	// Receiver & Transmitter enable: RXEN0,TXEN0
```

```
	UCSR0B = (1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0);

	// Along with UCSZ02 bit in UCSR0B, set 8 bits

	UCSR0C = (1<<UCSZ01)|(1<<UCSZ00)|(1<<UPM01)|(1<<USBS0);

	DDRD |= (1<< 1);	// PD0 is output (TX)
	DDRD &= ~(1<< 0);	// PD1 is input (Rx)

	// Empty buffers

	rx_buffer_head = 0;
	rx_buffer_tail = 0;
}


void usart_printf(const char *ptr){

	// Send NULL-terminated data from FLASH.
	// Uses polling (and it blocks).

	char c;

	while(pgm_read_byte_near(ptr)) {
		c = pgm_read_byte_near(ptr++);
		usart_putc(c);
	}
}

void usart_putc(const char c){
	// Send "c" via the USART.  Uses poling
	// (and it blocks). Wait for UDRE0 to become
	// set (=1), which indicates the UDR0 is empty
	// and can accept the next character.

	while (!(UCSR0A & (1<<UDRE0)));
	UDR0 = c;
}

void usart_prints(const char *ptr){
	// Send NULL-terminated data from SRAM.
	// Uses polling (and it blocks).
```

```
        while(*ptr) {
                while (!( UCSR0A & (1<<UDRE0)));
                UDR0 = *(ptr++);
        }
}

unsigned char usart_getc(void)
{
        // Get char from the receiver buffer.  This
        // function blocks until a character arrives.

        unsigned char c;

        // Wait for a character in the buffer.

        while (rx_buffer_tail == rx_buffer_head);

        c = rx_buffer[rx_buffer_tail];
        if (rx_buffer_tail == RX_BUFFER_SIZE-1)
        rx_buffer_tail = 0;
        else
        rx_buffer_tail++;
        return c;
}

unsigned char uart_buffer_empty(void)
{
        // Returns TRUE if receive buffer is empty.

        return (rx_buffer_tail == rx_buffer_head);
}
```