Shivam Desai
Asa Eckert-Erdheim
ENGR3199: Elecanisms
October 7, 2013

Mini-project 3: Ultrasonic Ranging System

All source code for this mini-project can be found on Asa and Shivam's Github repository called MiniProject3_Ultrasonic.

Introduction

An ultrasonic ranging system measures distance to target objects by emitting brief pulses of ultrasound and measuring the time it takes for any echoes to be received. The speed of sound in the propagation medium and time for the sound to make the round trip to a target is used to compute the distance to a target.

In this lab, we constructed an ultrasonic ranging system using our 2 DOF mechanical pan-tilt system from Mini-project 2. We constructed an ultrasonic transducer PCB, designed an ultrasonic transmitter and receiver circuit, calibrated our system, and extended our PIC/Python software to map the distance to targets in the full field of view of our gimbal system. Our system has a usable range of 30 – 200 centimeters. Objects nearer than 30cm are too close to differentiate between and objects farther than 2m are too far to get a clean/reliable reading.

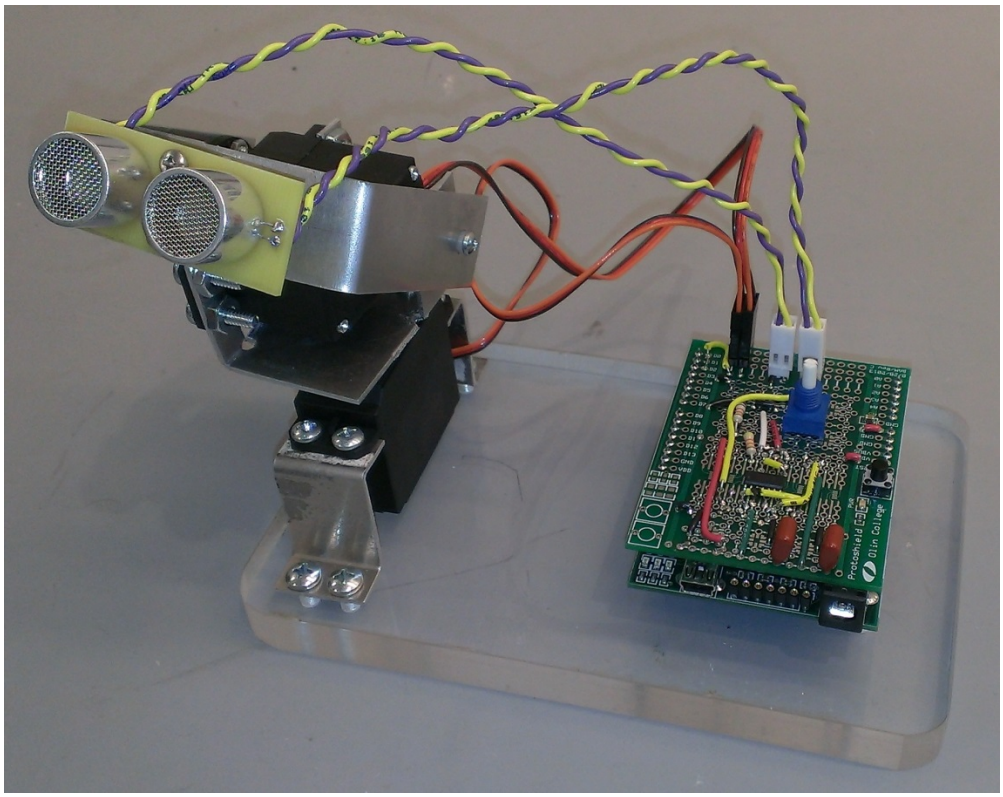

Figure 01: Fully assembled ultrasonic system with Elecanisms board

Electrical System

We soldered two Prowave 400SR160 ultrasonic transducers to a PCB and mounted them on the head of our gimbal system from Mini-project 2.

The 400SR160 ultrasonic transducers are tuned to send and receive ultrasound at 40 kHz, so our transmitter circuit drives the emitter at this frequency by sending a 40 kHz PWM signal with a duty cycle of 50%.

Considering that the received signal is very small and noisy, we wanted to implement a third-order band-pass filter centered at 40 kHz with a high gain (excess of a few thousand). We chose the band-pass filter shown below in Figure 02.

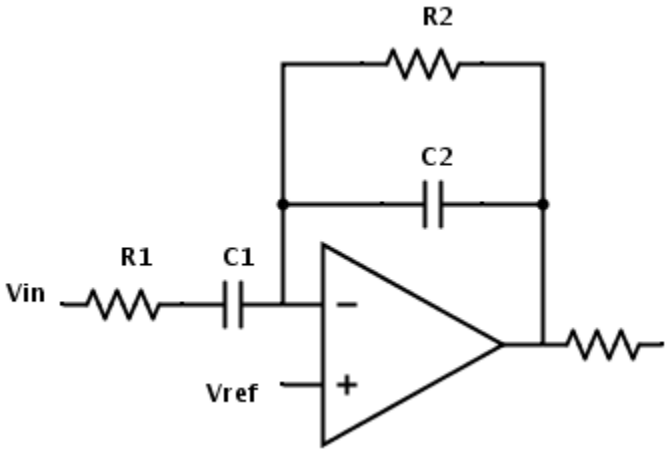

Figure 02: Band-pass filter used

With V_{ref} set at ground, we can use ohm's law and Kirchoff's current law to arrive at the equation below.

$$\frac{V_{in}}{Z_1} = \frac{-V_{out}}{Z_2}$$

We can simplify to obtain the transfer function

$$\frac{V_{out}}{V_{in}} = \frac{-Z_2}{Z_1} = \frac{-(R_2 || \frac{1}{C_2 s})}{R_1 + \frac{1}{C_1 s}}$$

And simplify further to obtain

$$\frac{V_{out}}{V_{in}} = \frac{-R_2}{R_1} \times \frac{R_1 C_1 s}{R_1 C_1 s + 1} \times \frac{1}{R_2 C_2 s + 1}$$

Where the gain is given by $\frac{-R_2}{R_1}$, the high-pass component is given by $\frac{R_1C_1s}{R_1C_1s+1}$, and the low-pass component is given by $\frac{1}{R_2C_2s+1}$.

We selected $R_1 = 604\Omega$, $C_1 = 10nF$, $R_2 = 30.1k\Omega$, and $C_2 = 100pF$ so that we could use the surface mount components provided to us and our pass band would be between 26 kHz and 52 kHz as seen below.

$$\frac{1}{2\pi R_1 C_1} = \frac{1}{2\pi \times 604\Omega \times 10nF} = 26.35\ kHz$$

$$\frac{1}{2\pi R_2 C_2} = \frac{1}{2\pi \times 30.1\ k\Omega \times 100pF} = 52.875\ kHz$$

At each stage our gain was $\frac{-R_2}{R_1} = \frac{-30100}{604} = -49$. Considering that we wanted to amplify our signal substantially with a third-order roll-off, we stacked three of these filters in series for a total gain of -123762 as seen in Figure 03.

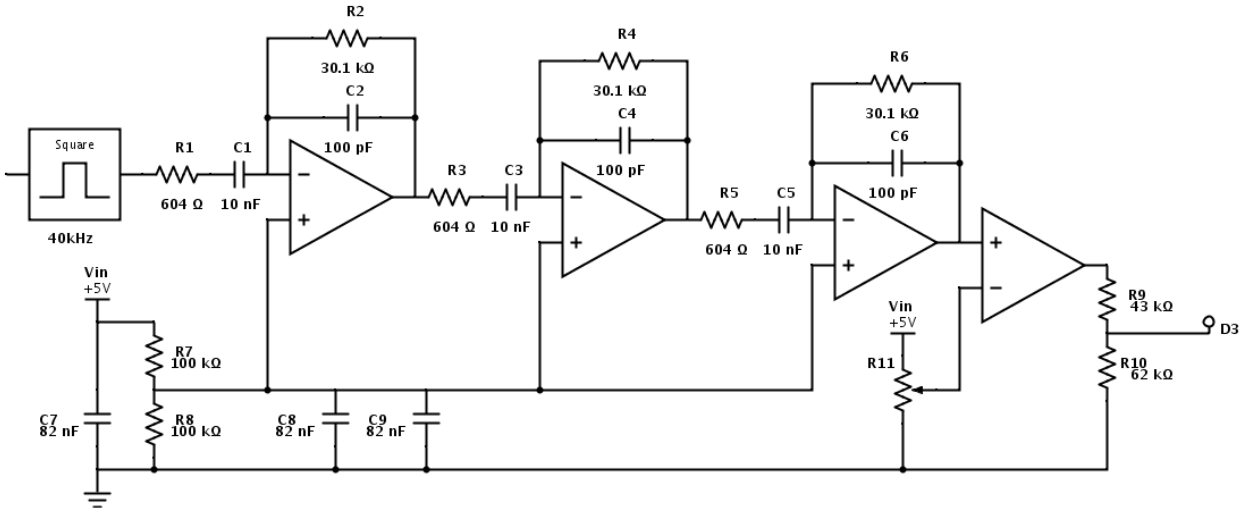

Figure 03: Schematic for Ultrasonic Receiver

After the filter chain, we threshold the final signal and use a comparator to provide a nice digital signal for the PIC. In order for the digital signal to wire directly into the PIC, we scale down the input voltage with a voltage divider. We also used multiple bypass capacitors of $82\ nF$ to remove noise from the system.

We implemented our design on the protoshield from Mini-project 2 with a combination of through-hole components and surface-mount (SMT) components as seen in Figure 04. Unfortunately, we were unable to remove a 400 Hz noise signal from the protoshield circuit, so we proceeded with a working breadboard implementation.

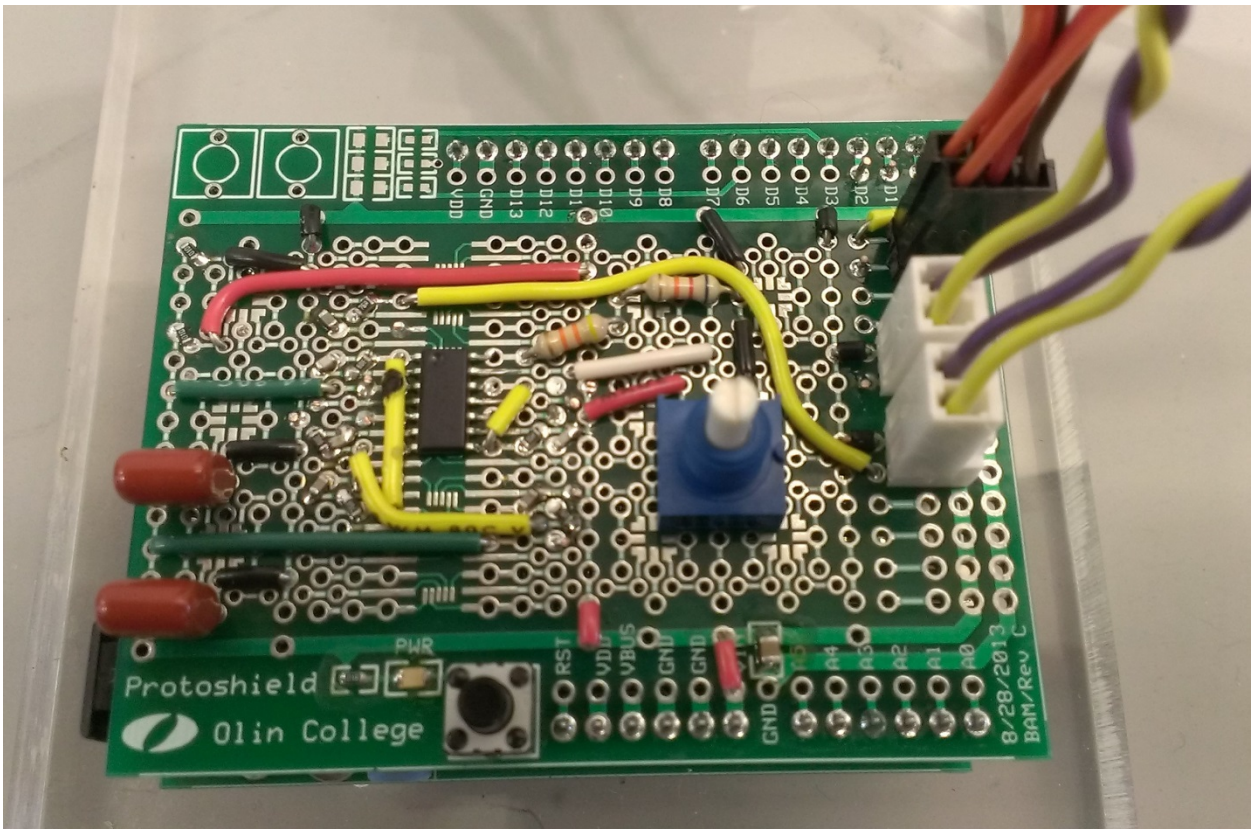

Figure 04: Implementation of ultrasonic circuitry on Elecanisms protoboard

Software

There are two components in the software system: embedded code running on the elecanisms board and host software running on a computer. The embedded code is written in C and the host software is written in Python. At a high level, the code works as follows: the user initializes the system using the host software, the host software sends commands to the PIC to set servo position and pings the ultrasonic sensor via USB, the embedded code executes said commands and returns values via USB, and finally the host software processes this data and plots it.

Embedded Software

The embedded software is very similar to that written for Mini-project 2, except for the addition of the ping ultrasonic behavior. This behavior creates the ultrasonic ping and measures the time it takes for the signal to return. To create the ultrasonic ping, we use the oc module to generate a 40 kHz PWM signal. The pulse time of the signal was set using a while loop – we set the PWM signal to a duty cycle of 50%, waited for a timer set to 500 microseconds to overflow, then set the duty cycle to 0%. Ideally, we would then be able to start looking for a return signal. However, we observed a significant amount of crosstalk between the transmitter and reciever. Our solution for this was to wait a fixed period of time after the pulse ended before looking for a received signal. This was implemented by counting the timer ticks the pulse was on, then waiting 4 times the pulse time after the pulse before looking for a receive signal. There are two important notes about this - timer ticks was chosen over timer time because timer time

had inconsistant behavior (we assume that this behavior happened because the PIC does not have float support and timer time is a float) and this wait effectively sets a minimum measureable distance. Any distance less than ~30cm will be ignored in our implementation.

Once the code delayed for an appropriate amount of time, it enters a while loop and waits for a receive signal. There are three possible outcomes to this: a signal is received in t < overflow time, a signal is received in t > overflow time, and a signal is never received. In our initial code, there was no differentiation between the first two cases. This led to a number of problems where farther distances appeared closer. To account for overflow, we created an overflow counter which tracks the number of overflows. The function times out if a return signal is never received. Due the previously mentioned problems surround timer time, we used an alernate method of tracking elasped time rather than timer time. Specificially, we waited for 20 timer overflows to occur before determining there was no return signal.

Once a signal has been receieved (or not), the time of flight (measured in ticks) and number of overflows is written via USB to the computer using code identical to that implemented for the provided get_vals case.

Host Software

The primary functionality of the host software is to write and receive data to/from the Elecanisms board. However, there is a substantial amount of supporting code that ended up comprising most of the host software. This mostly includes calibration and plotting.

The calibration code is used to relate time of flight to measured distance. This is accomplished by measuring the time of flight at a number of distances and calculating the average speed. Specifically, we used 15 logarithmically spaced distances ranging from 30cm to 200cm. The calibration curve resulting from this is shown in Figure 04. Using this method, we calculated the speed of the signal (the speed of sound) to be 0.0011 centimeters per tick.

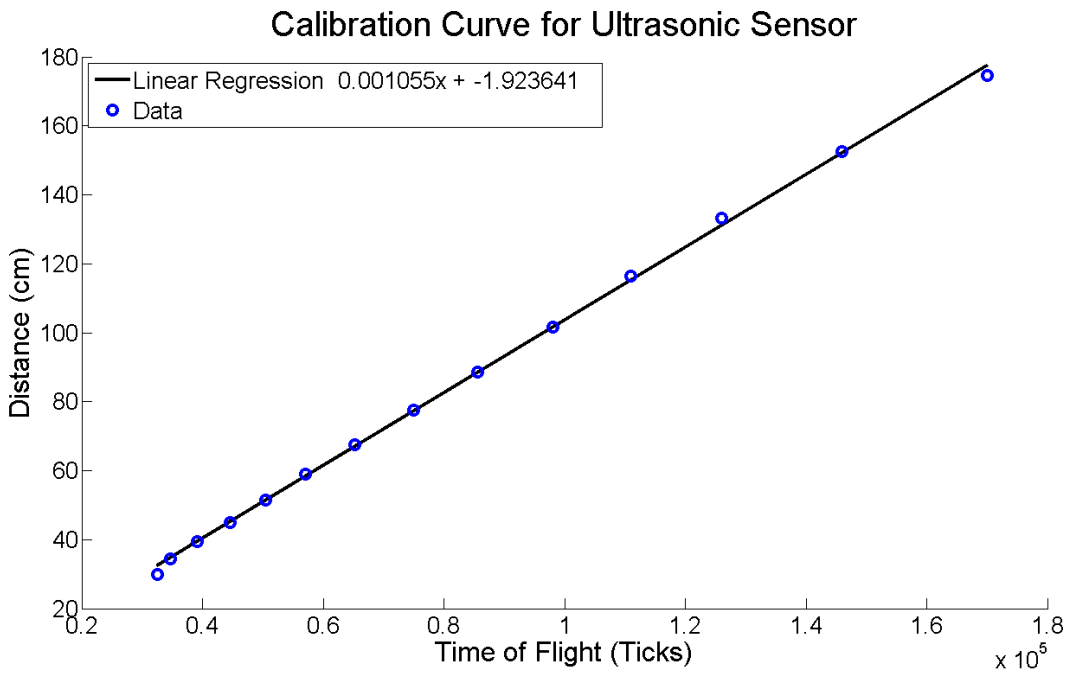

Figure 04: Calibration curve for ultrasonic range finder. The slope of the linear regression is the measured speed of sound in centimeters per tick.

Once calibrated, the main host loop is run. This consists of sending command to move the servo and sending/receiving signals from the ultrasonic sensor. The time of flight received is converted to a measured distance using the previously measured signal speed.

At this point, the sensor data exists in spherical coordinates: the servo angles and measured distance. In order to visualize the data, this data is converted to cartesian coordinates using the following equations. Note that these equations have been shifted 90° to account for the tilt servo angle being with relation to the horizontal plane rather than vertical.

$$X = d \cos\theta \cos\emptyset$$
$$Y = d \sin\theta \cos\emptyset$$
$$Z = d \sin\emptyset$$

To visualize this data, a simple 3D point cloud is created using a matploblib 3D scatter plot. The code makes use of matplotlib animation to visualize the data in realtime. An example of a point cloud can be seen in Figure 05.

Final Performance

Using the circuit and software design presented in this report, we were able to create a simple ultrasonic range finder capable of mapping the area around it. One such map is shown below in Figure 05.

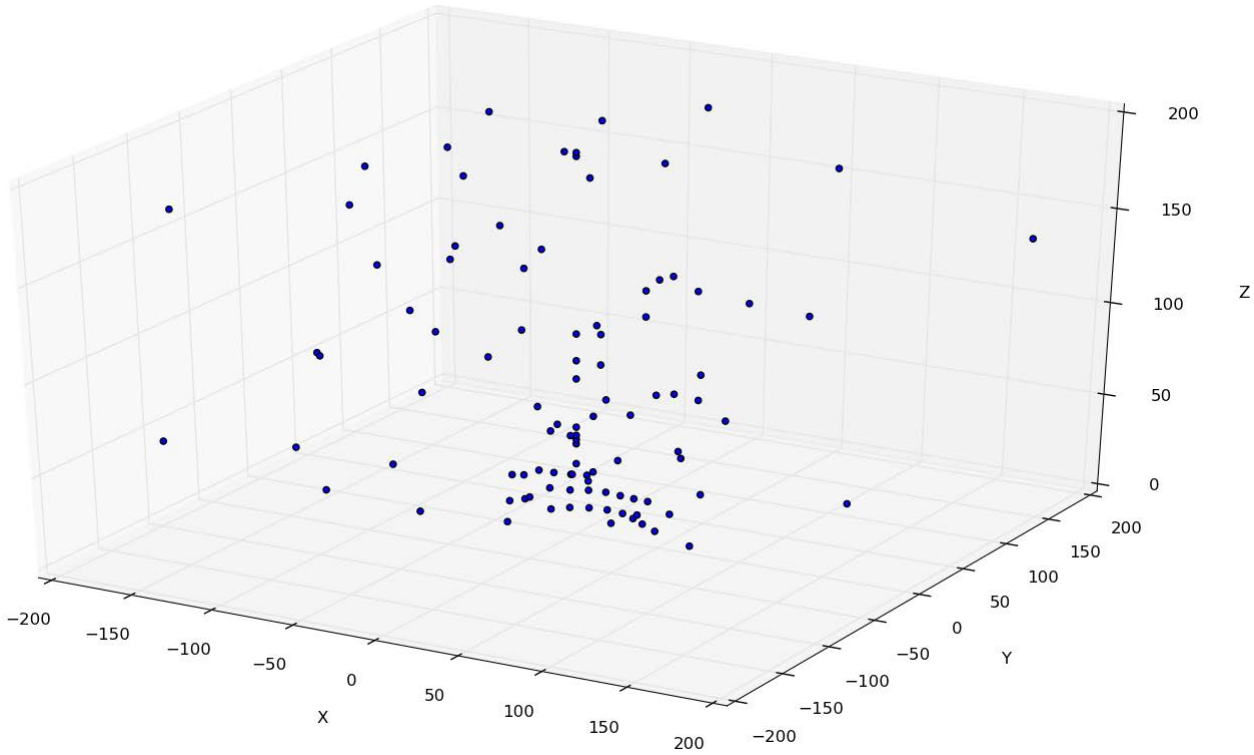

Figure 05: 3D point cloud of data generated by the ultrasonic range finder. This test was run with the sensor sitting on a table in the middle of Academic Center room #308. The two objects near the sensor are a laptop screen and a chair.

With this sensor, we are able to achieve a range of at least 2 meters. We are able to pick up objects beyond 2 meters, however, the amount of noise at that range makes getting a trustworthy reading quite difficult.