Sensors and Motors Lab
Individual Lab Report #1

Max Hu

Team Lo-Co
(Team D)

Teammates: Amit Bansal, Aum Jadhav,
Cyrus Liu, Kazuya Otani

October 14, 2016

Individual Progress

For this lab, I was primarily responsible for interfacing the ultrasonic sensor with the Arduino. My secondary role was to implement and tune the PID controller for the DC motor.

Ultrasonic Sensor

For the ultrasonic sensor, the first step to doing this was to identify the part number of the ultrasonic sensor begin used, and to obtain its datasheet online. The sensor provided is the LV-MaxSonar-EZ1 (MB1010). From the datasheet, I obtained some key information required, shown in Table 1.

Power Requirements	2.5V-5.5V (3mA for 5V, 2mA for 3V)
Data	Analog Voltage Out Serial Output (Tx/Rx) PWM Output

Table 1: Information about the LV-MaxSonar-EZ1

Since the current draw is very small, the sensor can be powered from the Arduino's 5V pin directly.There is an option of receiving data from the sensor by either reading the analog voltage from the AN pin, or through a serial connection through the Tx/Rx pins. I opted to read the analog voltage because it's much simpler and required only one data pin to be connected instead of two for serial.
Based on the above, I wired up the sensor to the Arduino according to the mapping showed in Table 2.

Arduino Pin Name	Sensor Pin Name
+5V	+5V (VCC)
GND	GND
A0	AN

Table 2: Pin Mapping

With the sensor connected, the next part is to figure out how to read it. The following information is obtained from the datasheet:

> AN- Outputs analog voltage with a scaling factor of (Vcc/512) per inch. A supply of 5V yields ~9.8mV/in. and 3.3V yields ~6.4mV/in. The output is buffered and corresponds to the most recent range data.

From the information provided in Figure 1, the formula for calculating the range output by the sensor in inches can be derived:

$$V_o = \frac{analogRead(A0)}{1024} * 5V$$
$$Range\ (in.) = \frac{V_o}{\frac{5V}{512}} = \frac{analogRead(A0)}{1024} * 5V * \frac{512}{5V} = \frac{analogRead(A0)}{2}$$

Hence, it turns out that the range in inches is approximately half of the analogRead(A0) value.

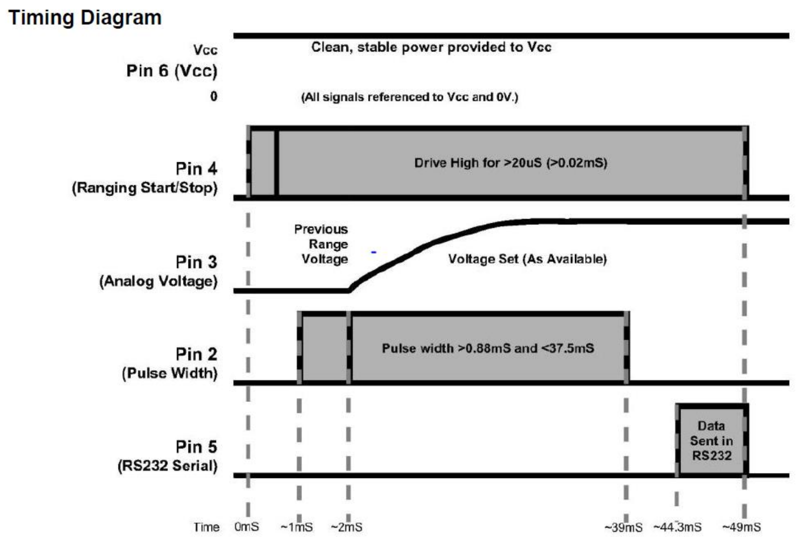


Figure 2: Timing Diagram for the ultrasonic sensor

The datasheet also provides a Timing Diagram, shown in Figure 2, which shows that the analog voltage takes some time to settle to the final value, so the ADC can be sampled at ~50ms intervals to ensure that the analog voltage output has settled before sampling it.
At this point, I was able to get an output of distance in inches, at a sampling rate of about 20Hz.

PID Control

The DC motor used is affixed with an incremental quadrature encoder, which we can use to implement closed-loop position control for the DC motor.

From the number of encoder ticks we are able to get an estimate of the position of the shaft. When a new desired position is received from one of the sensors, the controller computes the error, which is given by the following equation:

$$error, e(t) = position_{desired} - position_{current}$$

A PID controller is then used to compute the speed required to regulate the error to zero. A larger error in the shaft position will result in a greater control input to drive the shaft to the desired position at a greater speed. The errors are also accumulated and differentiated for the integral and derivative parts of the controller respectively. The time step is determined by using the millis() function at the start and at the end of the control loop.

Choosing appropriate proportional, integral and derivative gains is a delicate task. I started with a purely proportional controller by setting the K_i and K_d gains to 0, and increasing the K_p gain to a point where the rise time is fast enough, but with a little overshoot. The next step is to introduce a small K_d gain to reduce the overshoot and reduce the settling time. Once this is tuned to an appropriate amount, the K_i term is introduced to reduce the rise time, allowing for the motor to respond more quickly to rapid changes in the sensor values.

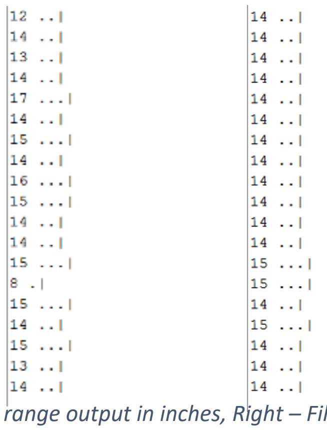
Challenges

Ultrasonic Sensor

The ultrasonic sensor, while easy to use, provided very noisy sensor data, with large fluctuations in the output value even when pointed at a flat, static surface.

To tackle this, I implemented a Median Filter so that the sensor provides smoother readings. Each ADC sample is first added to an array of an appropriate size, which in this case, an array size of 5 was chosen as it represents a good balance between filtering performance and sampling time. The readings are then sorted in an increasing order within the array, and the middle value is chosen to be the output.

The filter does a pretty good job of stabilizing the output value, but the sampling rate takes a hit from the original maximum of 20Hz to about 4Hz. Figure 3 compares the output between the filtered and unfiltered data.



```
12 ..|
14 ..|
13 ..|
14 ..|
17 ...|
14 ..|
15 ...|
14 ..|
16 ...|
15 ...|
14 ..|
14 ..|
15 ...|
8 .|
15 ...|
14 ..|
15 ...|
13 ..|
14 ..|
```

```
14 ..|
14 ..|
14 ..|
14 ..|
14 ..|
14 ..|
14 ..|
14 ..|
14 ..|
14 ..|
14 ..|
14 ..|
15 ...|
15 ...|
14 ..|
15 ...|
14 ..|
14 ..|
14 ..|
```

Figure 3: Left – Unfiltered range output in inches, Right – Filtered range output in inches

PID Control

One challenge faced in this section was that the Arduino was reading the encoder pins too infrequently, and the shaft position could not be estimated at a fast enough rate. This was because the encoder pins were only polled once per loop. I switched to using an interrupt-based Encoder library, and that allowed the shaft position to be updated at a much faster rate.

Another challenge faced when controlling the DC motor is the ‘dead-band’, where the motor seems to be unresponsive when the sensor values are small. This happens because control inputs near the equilibrium point map to a range of voltages that are too low to drive the motor. I chose to tackle this problem via the software, by investigating the range of inputs whereby the motor stops responding, and clamping any input within the dead-band to the minimum input value required for the motor to be activated. Implementing this fix has significantly helped with the responsiveness of the motor when the sensor values are small.

Teamwork

- Amit Bansal: Servo motor, temperature sensor
- Aum Jadhav: DC motor w/ encoder, IR rangefinder, system integration
- Cyrus Liu: GUI
- Kazuya Otani: Stepper motor, potentiometer, hardware integration

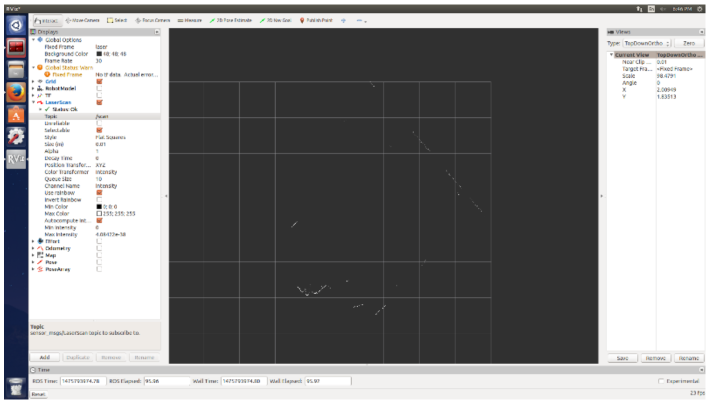
For this lab, our whole team of 5 started together at our team bench, with clearly defined tasks and responsibilities. The fact that we were working together meant that any questions or doubts can be resolved instantly, which would not have been possible if we had been working independently. This allowed us to complete the interfacing of individual sensors/motors in one afternoon.

Kazu was in charge of hardware integration, and wired up all the motors and sensors, while Aum took care of combining the code from each member. This allowed for the hardware platform with all the sensors/motors to be ready for testing very quickly. All this time, Cyrus had been working on the GUI, which was ready to be interfaced with the Arduino by the time the combined Arduino code was tested to be working.
What remained was purely software tuning of the control loops, which was a task undertaken by myself and Aum.

Future Plans

For the next few weeks, our team plan to take steps in getting the RC car platform up and running as soon as possible. We have already placed the order for several key components, and some of them have arrived.

At the same time, we have been working on hardware that we already possess, such as the Odroid XU4 SBC. I have been successful in setting up ROS on the SBC, interfacing it with a Hokuyo LIDAR, and visualizing the point cloud on a remote workstation on the same network. The 2D point cloud received from the LIDAR, as well as the hardware used, are shown in Figure 4. The next step will be to interface the Razor IMU with the SBC to get a working AHRS, followed by trying out some packages of the ROS Navigation stack to get localization working.



```
Global Options
  Fixed Frame            laser
  Background Color       48; 48; 48
  Frame Rate             30
Global Status: Warn
  Fixed Frame            No tf data.  Actual error...
Grid
RobotModel
TF
LaserScan
  Status: Ok
  Topic                  /scan
  Unreliable
  Selectable
  Style                  Flat Squares
  Size (m)               0.01
  Alpha                  1
  Decay Time             0
  Position Transfor...   XYZ
  Color Transformer      Intensity
  Queue Size             10
  Channel Name           intensity
  Use rainbow
  Invert Rainbow
  Min Color              0; 0; 0
  Max Color              255; 255; 255
  Autocompute Int...
  Min Intensity          0
  Max Intensity          4.08422e-38
Effort
Odometry
Map
Pose
PoseArray

Topic
sensor_msgs/LaserScan topic to subscribe to.

ROS Time: 1475793974.78   ROS Elapsed: 95.96
```

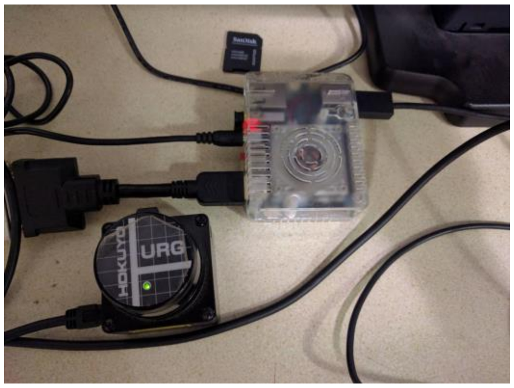


Figure 4: Left – Visualization of 2D point cloud from the Hokuyo LIDAR, Right – the Odroid XU4 SBC and the Hokuyo LIDAR

While waiting for the hardware components to come in, we will also be reading up on vehicle modelling and dynamics to try to develop a simple kinematic model for our system, so that we can begin simulating simple motions in a simulation environment. Once we get familiar with this, we can start introducing more complex elements into the vehicle model for simulation, which will help us with eventually developing a controller for the RC car.