Sensor Data Acquisition Robotic Arm to Demonstrate Implementation of Force, Temperature and Strain Sensors

Submitted to:

Dr. Chang-Hee Won

Final Project for ECE 8110

November 16, 2011

Prepared by:

Firdous Saleheen, Amrita Sahu and Tim Boger

Contents

Abstract

A sensing system capable of detecting the applied force, temperature and strain and quantifying it was designed, tested and implemented. Circuit setups were prepared for the three sensors. The circuit setup is then integrated with Arduino board for data acquisition. Manufacturer-provided voltage vs force curve and Matlab curve fitting tool was used to find the relation between output voltage of the circuit and applied force. Similarly, for temperature and strain calculations, manufacturer provided voltage curves were used to find the temperature and strain values.To test the performance an experiment, plastic balls of different elasticity and fruits of different hardness were used. The experimental result showed that the system can detect the applied force , temperature of the balls and also the strain when the ball is deformed, and measure it. The long term goal of this project to have a sensing system that could be used in biomedical applications, particularly in the tactile imaging system, to get the strain, stress and temperature information of the tissue inclusions

Introduction

TB, AS, FS

The final project for the Sensors course was to implement a sensor data acquisition robotic arm for the purpose of demonstrating the implementation of applicable sensors including force, temperature and strain. The force, temperature and strain sensors were attached to the claw of the robotic arm and an Arduino Board was used to control the robotic arm autonomously and to perform data acquisition with each of the sensors. An ultrasonic sensor has also been used in this project. It provides a very low-cost and easy method of distance measurement. The Arduino Board can be programmed using Arduino IDE or Maltab. We have used both the programming environments in our project. Both allow the Robotic arm to be programmed in any desired manner. The basic program appended to this report sets up the Robotic arm to autonomously measure sensory inputs of a small rubber ball and display it serially. The purpose of this project was to learn about sensors and system integration by designing, building and testing a sensing system. Our sensing system has successfully integrated the following four sensors: Ultrasonic, force, temperature and strain gages. The long term goal of this project to have a sensing system that could be used in biomedical applications, particularly in the tactile imaging system, to get the strain, stress and temperature information of the tissue inclusions.

The project report is divided into several sections. Section 4 describes the materials used in the project and their total cost. In Section 5 the robotic arm implementation is discussed. Section 6 describes the operation of the ultrasonic sensor. Section 7 and Section 8 describes the implementation of the force and the temperature sensor respectively. Then in Section 9 the strain gage implementation is discussed. Then finally the report ends with the discussion and conclusion part.

Materials

Following is the list of the materials used in the project and the total project cost

Item	Description	Manufacture	Cost
Robotic Arm	Robotic Arm Kit	OWI	$39.91
Arduino	Microcontroller	Arduino	$28.80
Motor Shield	Controls DC Motors	Adafruit	$19.90
Ultrasonic Ping Sensor	Sensor that Pings sound waves to determine distance, etc.	Parallax	$29.99
Potentiometers	Used as encoders	Any	$5.00
Force Sensor	For gripper	Digikey	$8.11
Temperature Sensor	For gripper	Digikey	$ 11.00(pkg of 10)
Strain Gages	For gripper	Digikey	$28.80(pkg of 5)
Total			$171.51

Robotic Arm

Robot Arm Kit (Timothy, Amrita)

The robotic arm was a method to implement sensor acquisition testing. The robotic arm was constructed from a OWI Robotic Arm Edge kit Model OWI-535. The robotic arm is depicted in Figure 1.

Figure 1. OWI Robotic Arm Edge

The robotic arm sits at 6.3 x 15 x 9 inches and weighs 2.5 on its own. Its original design was powered by 4 D Batteries and controlled with a 5 switch hardwired controller. The robotic arm has 5 control points that allow for commanding the robotic arm gripper to open and close, radial wrist motion of 120°, an extensive elbow range of motion of 300°, base rotation of 270°, base motion of 180°, vertical reach of 15 inches, horizontal reach of 12.6 inches and lifting capacity of 100g. The basic overview of the robotic arm is shown in Figure 2.

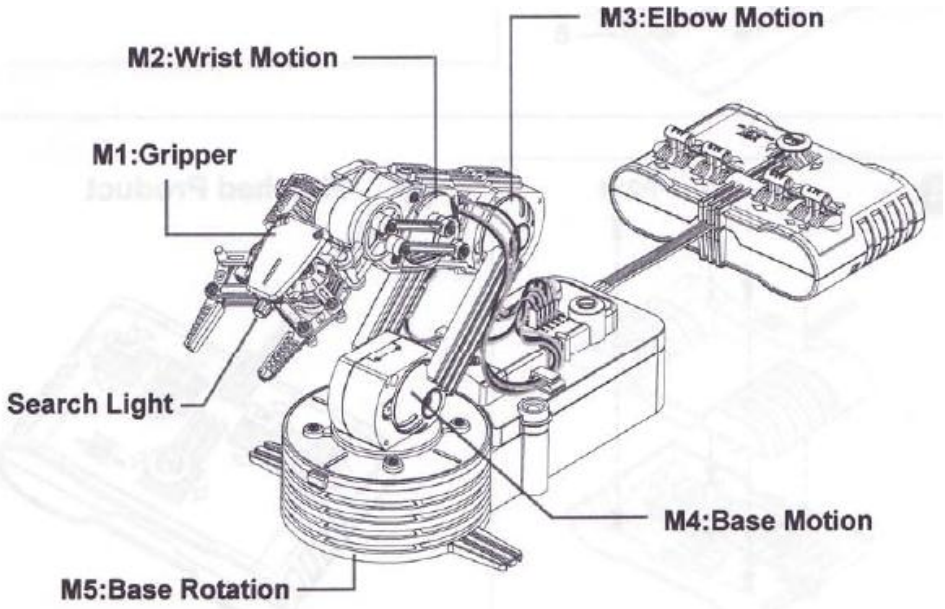


Figure 2. OWI Robotic Arm Edge Key Features

Some of the added features include a search light design on the gripper and an audible safety gear indicator which is included on all 5 gear boxes to prevent potential injury or gear breakage during operation. The five-switch wired controller, controls 5 motors contained within the 5 gear boxes that power the 5 joints of the robotic arm. The axis's of rotation and degrees of movement of the robotic arm demonstrated in Figure 3.

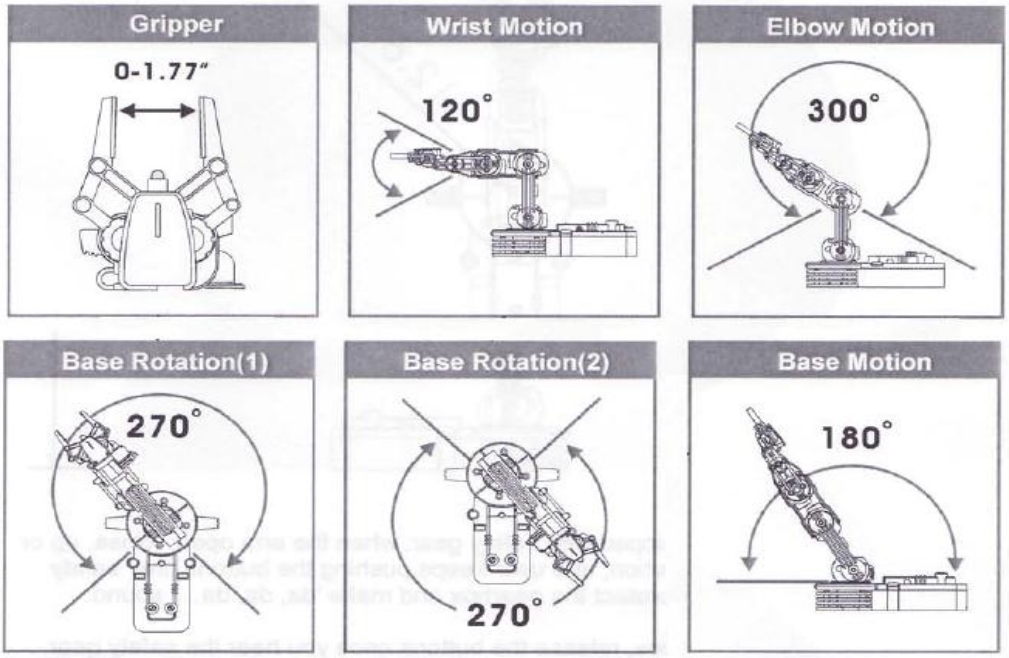


Figure 3. OWI Robotic Arm Edge Axis and Degrees of Rotations

The ranges of motion and reach capabilities of the robotic arm are shown in Figure 4.

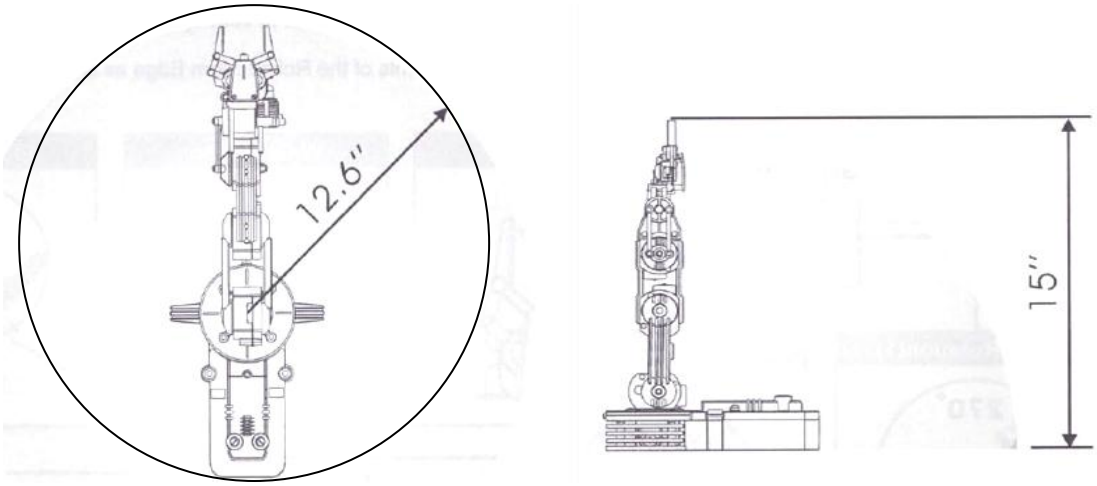


Figure 4. OWI Robotic Arm Edge Ranges of Motion and Reach Capabilities

The robotic arm comes unassembled and was contracted as designed by the manufacture. This robotic arm was chosen for its ability to be redesigned as desired. The wired diagram of the robotic arm is shown in Figure 5.

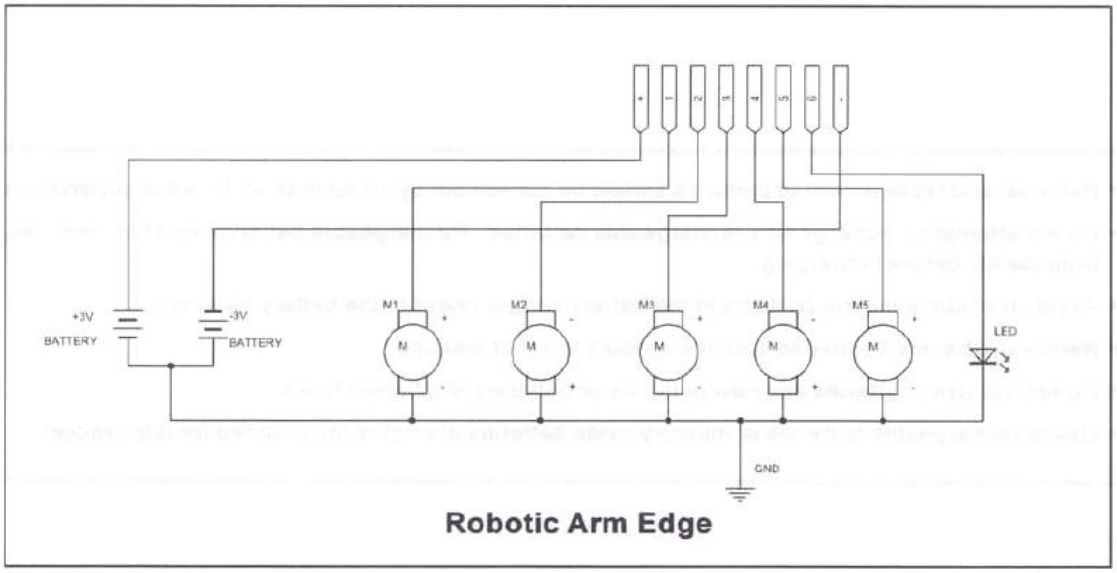


Figure 5. OWI Robotic Arm Edge Wire Diagram

All of the motor control wires can be easily removed and controlled with another desired hardware controller. The low cost of the robotic arm allowed for adjustments without damaging the functionality of the robotic arm. This limiting factor that controls the price of this and other robotic arms is the type of

motor used. This type of robotic arm is inexpensive due to it being controlled by DC motors and not servos. Figure 6 shows an example of a servo motor that is used in the majority of moderate robotic arms.

Figure 6. DC Motor Controlling Robotic Arm

The reason a servo motor requires little hardware is because all of it functionality is controlled through three wires, a 5v, GND, and Signal input wire. It is controlled with a PWM signal which can control direction and speed with a single signal wire. Using a DC motor makes the required controlling hardware a little more complex but it reduces the cost of the project. The type of DC motor controlling the robotic arm is shown in Figure 7.

Figure 7. DC Motor Controlling Robotic Arm

This type of motor requires the polarity of the wires to be switched in order to control the direction of the motor. This requires some type of additional switching hardware in order for the motors to be controlled bi-directionally.

Arduino Board (Timothy)

Controlling the robotic arm autonomously and to perform data acquisition with each of the sensors, a controller was needed. The Arduino Uno microcontroller module has a USB connection interface. The Arduino IDE and libraries are open source and there are a large variety of accessory, Shields, that drop onto the microcontroller. It has a variety of I/O pins including analog, digital, and PWM ports. The Arduino Uno microcontroller is shown in Figure 8.

Figure 8. Arduino Uno Microcontroller

The Arduino Uno can be powered through the USB connection or with an external power supply. The power source is selected automatically. External power can come either from an AC-to-DC adapter or battery. The Arduino Uno can be programmed with the Arduino software. Additionals specifications of the microcontroller are provided below.

Arduino Uno Specifications

- Microcontroller: ATmega328
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Input Voltage (limits): 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6
- DC Current per I/O Pin: 40 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB (ATmega328) of which 0.5 KB used by bootloader
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Clock Speed: 16 MHz

Dc motors can be controlled through the digital pins and the sensor data is acquired through the analog pins on the board. However additional hardware is still needed to control the DC motors bi-directionally. There are several motor shields designed for the Arduino Uno board.

Motor Shield (Timothy)

The DC motors of the robotic arm can be controlled by the Arduino Uno microcontroller using a motor shield. The Adafruit Motor Shield Kit for Arduino is a cheap unassembled kit that needs to be soldered together and allows for additional soldering as needed. The assembled Adafruit Motor Shield is shown in Figure 9.

Figure 9. Adafruit Motor Shield Sitting on Arduino Microcontroller

The motor shield has the capable of controlling the following:

- 2 connections for 5V servos
- 4 H-Bridges: L293D chipset provides 0.6A per bridge (1.2A peak) with thermal shutdown protection, internal kickback protection diodes.
- Can run motors on 4.5VDC to 36VDC.
- Up to 4 bi-directional DC motors with individual 8-bit speed selection
- Up to 2 stepper motors with single coil, double coil or interleaved stepping.
- 2-pin terminal block and jumper to connect external power, for separate logic/motor supplies
- Dimensions: 2.7in x 2.1in x 0.6in
- Weight: 32g

There are two provided library that allow for easy control of the various motors. The two libraries are the AFMotor.h and ServoTimer1.h library files, however the AFMotor.h is the only one needed for this project. The layout of the motor shield is shown in Figure 10. The motor shield's limit is four DC motors, however the robotic arm has 5 dc motors to control. For this specific project, the rotation of the base is not needed and is ignored. The other four motors can be controlled using the for motor ports depicted in Figure 10. A external power supply to power the motors is needed because the USB cannot supply enough power to move multiple motors at once. The external power supply connection is also depicted. The motor shield occupies several of the Arduino board pins, but leaves digital pins 2, 9, 10, and 13 and the 6 analog pins A0-5 to be used for running other desired hardware. If needed, 5 volts can also be taken from the Arduino board to power additional hardware.

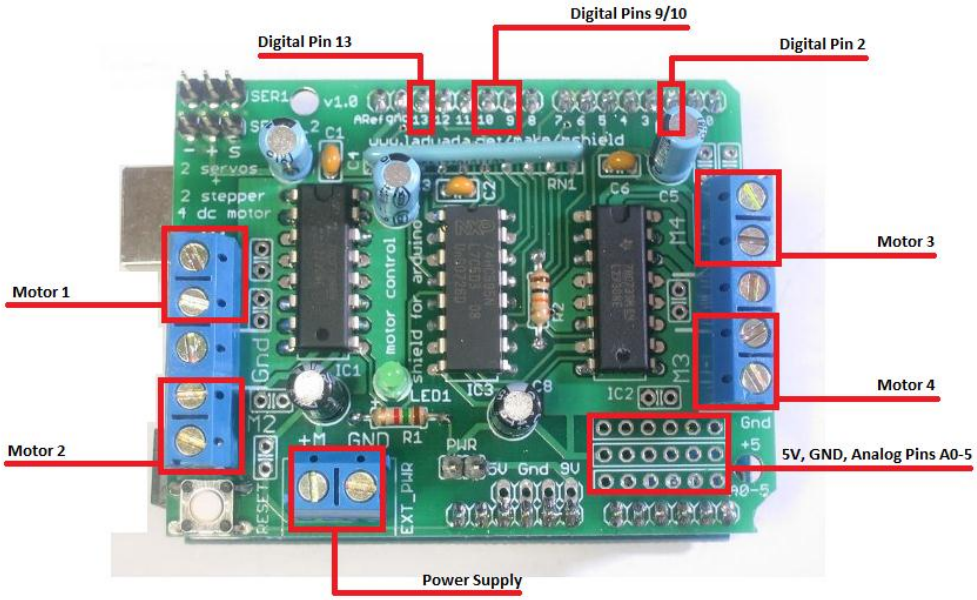


Figure 10. Adafruit Motor Shield Layout

The motor shield needs to be setup to use the external power supply. The wire diagram do so is shown in Figure 11. The jumper for the motor shield needs to be removed in order to use the external power supply.

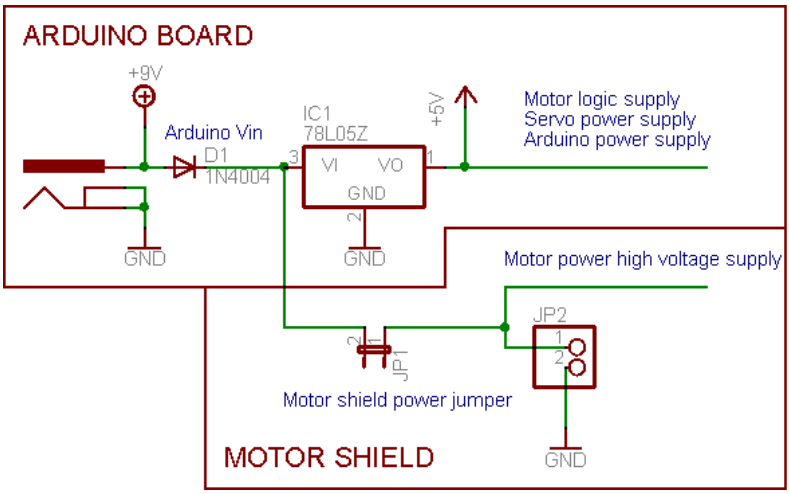


Figure 11. Adafruit Motor Shield Power Supply Wire Diagram

Limit Switches (Timothy, Amrita)

Another problem that arises when using dc motors over servos is being unable to control or monitor precise motor changes. Even if a dc motor is ran in one direction for a set duration and then ran the opposite for the same duration, it would not return to its original location. This is due to the mechanics of the dc motor. Therefore so type of encoder or sensor is needed to track the robotic arms position. If an

encoder is not used, the Arduino has no way of monitoring motor changes for closed loop feedback. A potentiometer can be used to encode the robotic arms location by providing a ground and five volts to it and monitoring the variance through a digital pin on the Arduino. The potentiometer used is shown in Figure 12.

Figure 12. Potentiometer Used for Robotic Arm Encoding

The potentiometer can be mounted to a stationary area near the pivoting section and the dial becomes attached to the moving arm using a bracket. The variance is then inputted into the microcontroller.

Integrating the Robot (Timothy)

Each hardware component, once prepared, was integrated together. The system block diagram is shown in Figure 13.

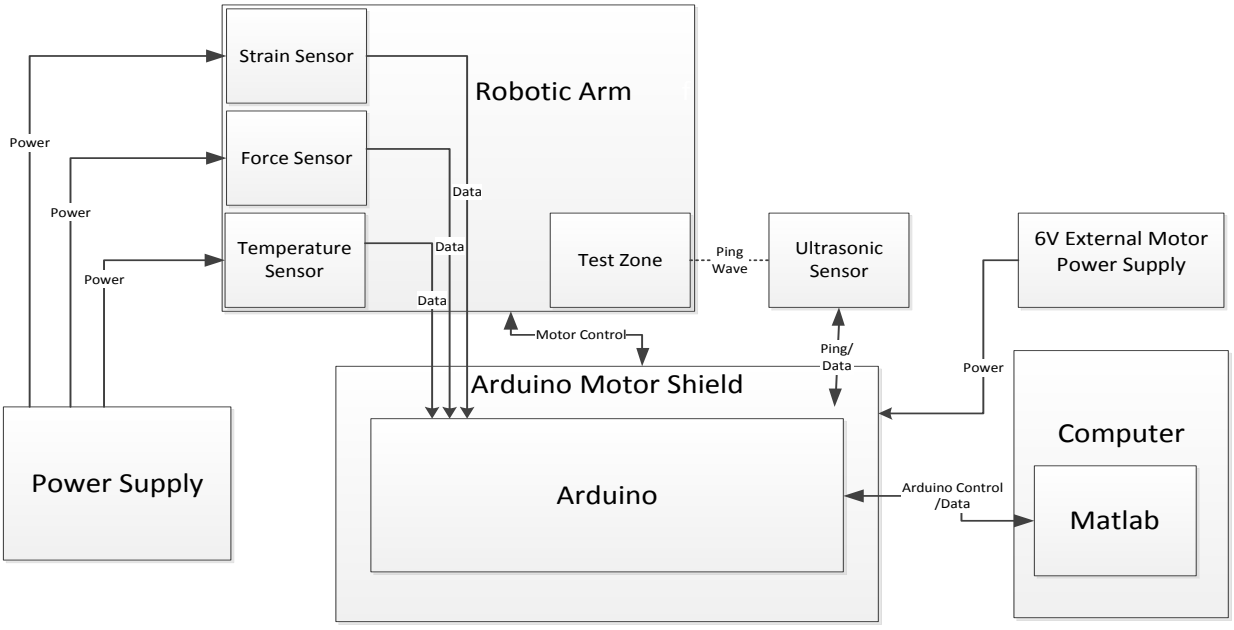


Figure 13. System Block Diagram

The Robotic Arm holds all three sensors and has a Test Zone where the object the sensors are acquiring data from is placed. The three sensors, the force, temperature, and strain, required additional circuitry to operate and have separate power supply to prepare their measurements, amplification, etc. for the Arduino board to accept as an input. The motor shield controls the dc motors of the Robotic Arm and the potentiometers send feedback data to the Arduino board to report the robotic arms current location. The ultrasonic monitors the test location and senses when an object is ready to be tested by the Robotic Arm by pinging the test location. A 6v external power supply is used to power the dc motors of the Robotic Arm since the USB connection cannot provide enough power to operate simultaneous dc motor operations. Finally, a computer is serial communicating with the Arduino board to accept the sensor data passed to it using the Arduino microcontroller to be analyzed in Matlab.

Programming (Timothy)

The Arduino board is programmed using the Arduino IDE. The environment is depicted in Figure 14.

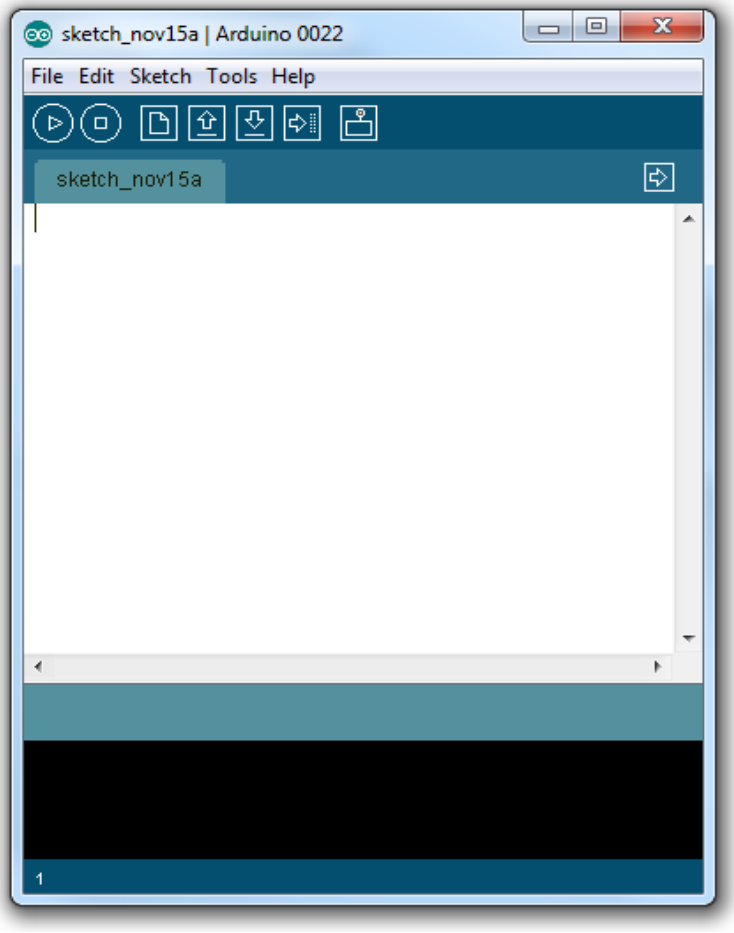


Figure 14. Arduino IDE

The Arduino board can also be directly programmed through Matlab. Both allow the Robotic arm to be programmed in any desired manner. The basic program appended to this report sets up the Robotic arm to autonomously measure sensory inputs of a small rubber ball and display it serially. When the program is uploaded, the robotic arm waits for a rubber ball to be placed in front of the test area. The ultrasonic acknowledges an object has been placed in front of it and initializes the robotic arm to tilt down, grip the rubber ball for 2 seconds to measure sensory data, and then realizes its grip and return to its home position. Figure 15 shows the Arduino controlling the robotic arm and Figure 16 shows the integration of the force sensor.

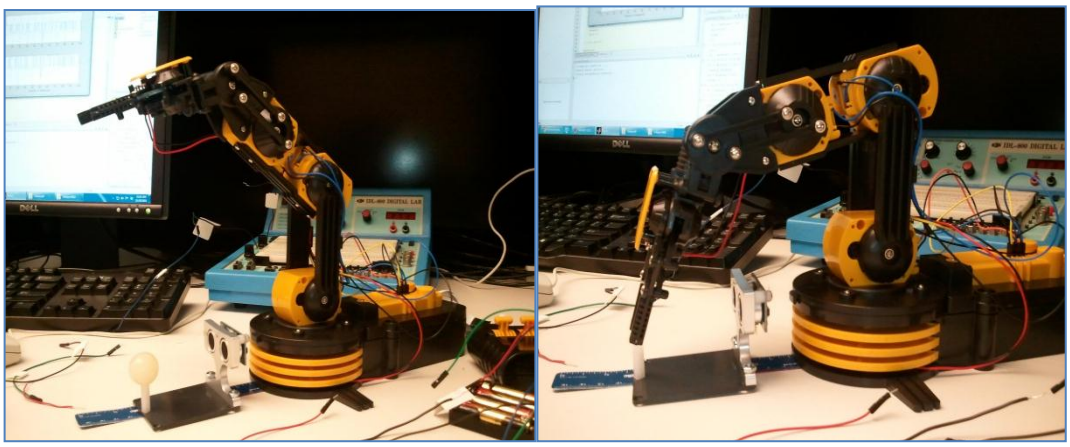


Figure 15. Arduino Controlling the Robotic Arm with Ultrasonic Sensor Only

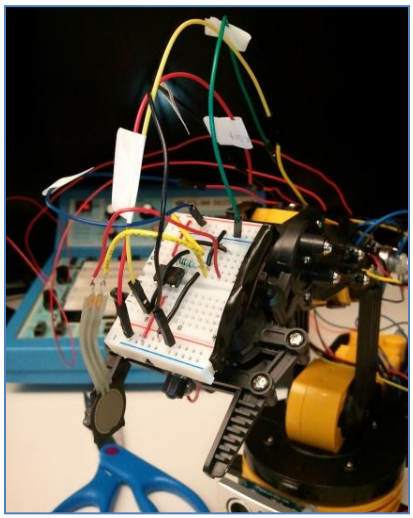


Figure 16. Integration of Force Sensor

Ultrasonics (Timothy, Amrita)

The Parallax's PING)))™ ultrasonic sensor provides a very low-cost and easy method of distance measurement. The Ping sensor measures distance using sonar. It provides distance measurements of a 2 cm to 3 m range. It has a power requirements of 5 volts with dimensions: 0.81 x 1.8 x 0.6 in and an operating temp range: +32 to +158 °F. The sensor is depicted in Figure 17.

Figure 17. Ultrasonic Ping Sensor and Stand

Force Sensor (Firdous)

A force sensing system capable of detecting the applied force and quantifying it was designed, tested and implemented. A circuitry for calculation of force applied on force sensing resistor has been designed, simulated and implemented on breadboard. The circuit setup is then integrated with Arduino board for data acquisition. Manufacturer-provided voltage vs force curve and Matlab curve fitting tool was used to find the relation between output voltage of the circuit and applied force. To test the performance an experiment, plastic balls of different elasticity and fruits of different hardness were used. The experimental result showed that the system can detect the applied force and measure it. From the measured force, stress (force per unit area) can be calculated which is one parameter to calculate Young's modulus or hardness

A force sensor measures applied force on some arbitrary target. For this purpose, a force sensing resistor is used. A force-sensing resistor (FSR) is a material whose resistance changes when a force or pressure is applied (1). It is of small size, low cost and easy to use.

System Design and Force Sensing Principle

System Component

The system components are:

1. Force Sensing Resistor
2. Measurement Circuit
3. Arduino Board
4. Matlab

Force Sensing Resistor

Model 402 round force sensing resistor made by Interlink Electronics was used. The active force sensing iameter is 12.7 millimeter.

Circuit for Force Measurement

Design

For a simple force-to-voltage conversion, the FSR device is tied to a measuring resistor in a voltage divider configuration. The output is described by the equation:

$$Vout = \frac{R1 * Vin}{R1 + R3}$$

In figure 1, the circuit for force measurement is shown where a pull-down resistor R1= 100k ohms, Vin = 5V is used. A voltage op-amp (uA741) follower is used to match the impedance requirement of the upstream measuring circuit. Vout is the output voltage measured at pin 6 of the op-amp. With this configuration, the output voltage increases with increasing force.

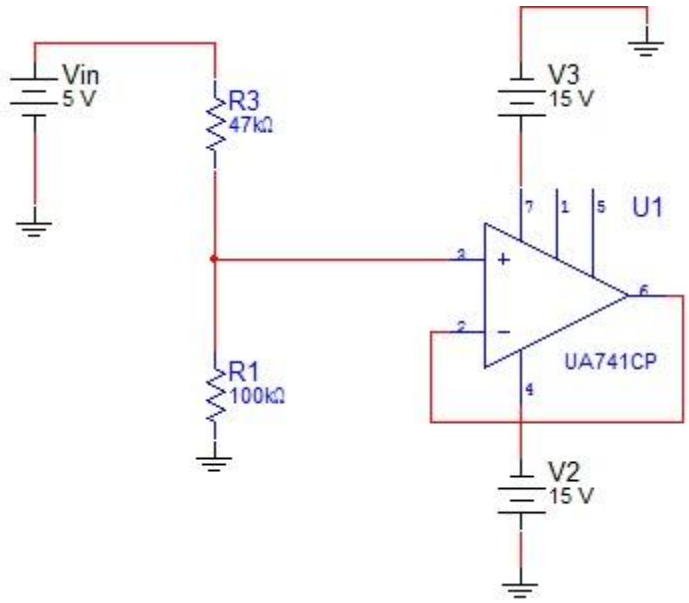


Figure 18: Voltage Divider Circuit for Force Sensing

Simulation

Transient analysis of the circuit from figure 1 has been done. Analysis shows the circuit gives desired output.

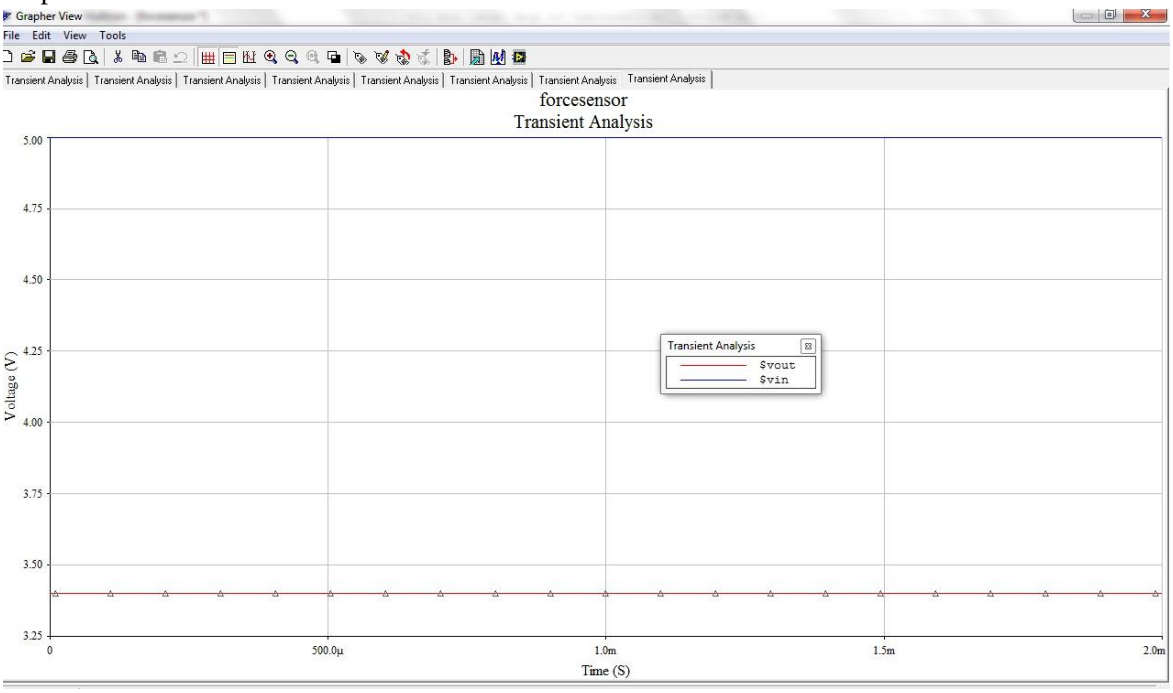


Figure 19: Simulation for circuit of figure 1

Implementation
The circuit has been implemented in a breadboard.

Voltage and Force Relationship
For calculation of force from Vout, this relationship has been used:

force = exp(log((voltage-6835)/(-2617))/(-0.1054))

The relationship has been derived from Manufacturer's data and using matlab curve fitting tool. See the detailed derivation in Appendix B – Curve fitting.

Integration with Arduino Board and Data Acquisition

Integration with Arduino Board
The following materials are required and have been used for integration of force sensor with Arduino Board:

1. Matlab 2010b
2. Arduino Uno board (http://arduino.cc/)
3. Arduino IDE
4. Matlab support package for arduino known as ARDUINOIO
5. Force Sensor with circuit

Steps for integration

1. Downloading and installing the IDE (to be done only once).
    a. A step by step driver installation can be found at: http://arduino.cc/en/Guide/HomePage and there is no need to duplicate it here. It is a good idea to go through all the 9 steps.
    b. After you have installed the drivers, the motor shield library, and verified that the IDE can communicate with the Arduino then you can start using this package.
2. Downloading and installing the motor shield library (to be done only once):
    a. Download the motor shield library for the motor shield here:http://www.ladyada.net/media/mshield/AFMotor-08_12_2009.zip then uncompress it and stick the AFMotor directory into the arduino-00XX/libraries folder.
3. Upload srv.pde (or Adiosrv.pde) to the Arduino board (to be done only once):
    a. The srv.pde (or adiosrv.pde) is the "server" program that will continuously run on the microcontroller. It listens for MATLAB commands arriving from the serial port, executes the commands, and, if needed, returns a result. The following instructions are needed to upload the srv.pde file into the controller's flash memory. Note that if you don't have the motor shield and don't plan to use it, then you can upload the adiosrv.pde file instead (the instructions are the same, except for the folder location). As long as no other file is uploaded later, this step does not need to be repeated anymore, and the package can be used as soon as the board is connected to the computer.
    b. From the Arduino IDE, go to File > Open, locate the file srv.pde, (in the ArduinoIO/pde/srv folder) and open it. If a dialog appears asking for the permission to

create a sketck folder and move the file, press OK (this will create a srv folder and move the srv.pde file inside it.

c. Connect the Arduino, make sure that the right board and serial port are selected in the IDE, (Tools/Board and Tool/Serial Port) then select File -> Upload to I/O Board and wait for the "Done Uploading" message.
d. At this point the srv.pde file is uploaded and you can close the IDE, which is not needed anymore for the purpose of this package. Actually closing the IDE is suggested, so you can be sure that the serial connection to the arduino board is not taken by the IDE when matlab needs to use it. Note that the older files adiosrv.pde (IO pins only) and motorsrv.pde (motor shield only) are also still available as simplified versions for you to play around and create your own sketch versions. For older boards,
e. should you have any connection problems, it is suggested that you try to upload and use the adiosrv.pde file.

4. Final Preliminary Steps(to be done only once):
    a. On Windows 7 you should be able to run MATLAB as administrator by right-clicking on the MATLAB icon and select "Run as Administrator". This will allow the updated path to be saved.
    b. From MATLAB, launch the "install_arduino" command, this will simply add the relevant ArduinoIO folders to the matlab path and save the path.
5. Typical Usage:
    a. Make sure the board is connected to the computer via USB port, make sure you know which serial port the Arduino is connected to (this is the same port found at the end of the drivers installation step), and finally, make sure that the port is not used by the IDE (in other words, the IDE must be closed or disconnected), so that MATLAB can use the serial connection.
    b. From MATLAB, launch the command a=arduino('port') where 'port' is the COM port to which the Arduino is connected to, e.g. 'COM5' or 'COM8' on Windows, for Arduino versions prior to Uno) and make sure the function terminates successfully.
    c. Then use the commands a.pinMode, (to change the mode of each pin between input and output) a.digitalRead, a.digitalWrite, a.analogRead, and a.analogWrite, to perform digital input, digital output, analog input, and analog output.
    d. Consult the help of the files to get more information about their usage.
    e. Finally, use a.delete to delete the arduino object, (and free up the serial port) when the session is over.NOTE: Should the serial port not be relaesed after you clear the arduino object, you can use the following commands to release serial connections.
    f. % delete all MATLAB serial connections delete(instrfind('Type', 'serial')); % delete MATLAB serial connection on COM3. delete(instrfind({'Port'},{'COM3'}));

Data Acquisition Steps:

1. Assuming the above instructions are followed properly, following code should be run in matlab.
2. The code reads analogpin(0) of Arduino board and converts the serial data into voltage and then in force.
3. The code also graphs the real time discrete data for voltage and force.
    a. Matlab Code is given below

```
% Author :Firdous Saleheen
% date 11/14/2011
% read serial data from force sensing resistor and plot against datapoints
clear all;
close all;
clc;
s1 = arduino('COM4');                    %define arduino object
%s1.BaudRate=9600;                        %define baud rate
%%
%initialize data acquisition parameters
clear data;
numberofData = 100;
data = zeros(1,numberofData);
v = zeros(1,numberofData);
f = zeros(1,numberofData);
i = 1;

for i= 1:numberofData                %acquisition of 100 points
  % read analog data from sensor
   data(i) = s1.analogRead(0);
   v(i) = data(i)*(5000/1024);
 figure(1);
subplot(2,1,1);
title('Voltage and Force data plot');
 stem(i,v(i));
xlabel('Number of datapoints');
   ylabel('Voltage (mV)');
   drawnow;
   grid on;
hold on;
% force voltage relationship is found using manufacturers curve
% force in newton and voltage in mV
% force = exp(log((voltage-c)/a)/b) with a= -2617, b = -0.1054, c = 6835
   subplot(2,1,2)
   f(i) = exp(log((v(i)-6835)/(-2617))/(-0.1054));
   stem(i,f(i),'r');
   xlabel('Number of datapoints');
   ylabel('Force (N)');
   drawnow;
   grid on;
   hold on;
end


% close session

delete(s1)
```

Application Software

Avocado ripeness lookup table was made based on (2). The program is tested based on fictitious data.

Pseudo code

1. Take user input of force and strain data
2. Make force data into stress data by dividing the force with area
3. Find Young's modulus for each datapoint
4. Take the mean of Young's modulus
5. Design a look up table
6. Check the look up table for decision
7. Display a decision

Matlab Code

```
% avocadomodulus2.m
% Author Firdous Saleheen
% Date 11/14/2011
%%
clear all;
close all;
clc;

%   Take user input of force and strain data. force in kN
force = [0.010 0.020 0.030 0.040 0.050 0.060 0.070 0.080 0.090 0.100];

%   Make force data into stress data by dividing the force with area
diameter = 12.7e-3;
radius = diameter/2;
area = pi*radius*radius;
stress = force./area;
strain = [1e-3 2e-3 3e-3 4e-3 5e-3 6e-3 7e-3 8e-3 9e-3 10e-3];
%%

youngmod = stress./strain
Y = mean(youngmod)
%%
%   Check the look up table for theroshold
%   Design a lookup table for modulus of elasticity(kN/m^2) Y for avocado
%   Display a decision
if Y > 480024
   disp('Super hard, Green Avocado')
elseif Y >= 325110
   disp('Harvested within 5 days, Hard, Green Avocado')
elseif Y >=  115007
```

```
   disp('Harvested within 5 to 10 days, Medium Hard Avocado')
elseif Y > 73550
   disp('Harvested within 10 to 15 days, Medium Soft, Ripe Avocado')
elseif Y > 48116
   disp('Harvested within 15 to 20 days, Soft Avocado')
elseif Y <= 48116
   disp('Harvested within more than 20 days, Super Soft Avocado')

end
```

Test Code Result

Y = 7.8941e+004

Harvested within 10 to 15 days, Medium Soft, Ripe Avocado

Experimental Setup and Procedures

Experimental setup

1. For the experimental setup the following materials are required:
   - Matlab 2010b
   - Arduino Uno board
   - Arduino IDE
   - Matlab support package for arduino known as ARDUINOIO
   - Force Sensor with circuit
   - Avocado of different hardness
   - Plastic balls of different hardness
   - Robot arm

Procedure for ball hardness experiment

1. Matlab, Arduino IDE and ARDUINOIO package should be installed into the workstation.
2. Arduino Board is for controlling robot arm and reading serial data from sensor.
3. Through the program in IDE the robot arm should reach the target (avocado) and touch it with sensor installed in its claw.
4. The matlab code should be run now which will read the serial port and return the real time voltage and force data.

Procedure for Fruit ripeness check experiment

5. Matlab, Arduino IDE and ARDUINOIO package should be installed into the workstation.
6. Arduino Board is for controlling robot arm and reading serial data from sensor.
7. Through the program in IDE the robot arm should reach the target (avocado) and touch it with sensor installed in its claw.
8. The matlab code should be run now which will read the serial port and return the real time voltage and force data.
9. This data will be used in the application software matlab code as an input where it is converted to stress.

10. Strain will be calculated with the strain gage. Details will be found in strain gage section.
11. From there a mean value of Young's modulus will be calculated and displayed the decision based on the lookup table.

Experimental Results

Ball hardness check experiment has returned the graph of figure 3.

Figure 20: ball experiment sensor data

Future Work

Fruit ripeness experiment plan is given. The future task would be to do the experiment according to plan.

Temperature Sensor (Amrita Sahu)

In this project an analog temperature (LM35 Precision Centigrade Temperature Sensor) was used. An analog temperature sensor is a chip that tells us what the ambient temperature is. These sensors use a solid-state technique to determine the temperature. That is to say, they don't use mercury (like old thermometers), bimetallic strips (like in some home thermometers or stoves), nor do they use thermistors (temperature sensitive resistors). Instead, they use the fact as temperature increases, the voltage across a diode increases at a known rate. (Technically, this is actually the voltage drop between the base and emitter of a transistor. By precisely amplifying the voltage change, it is easy to generate an analog signal that is directly proportional to temperature.

Because these sensors have no moving parts, they are precise, never wear out, don't need calibration, work under many environmental conditions, and are consistent between sensors and readings. Moreover they are very inexpensive and quite easy to use.

Measuring temperature using LM35 Precision Centigrade Temperature Sensor

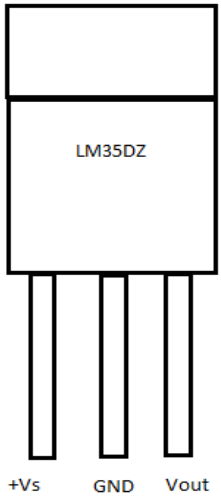
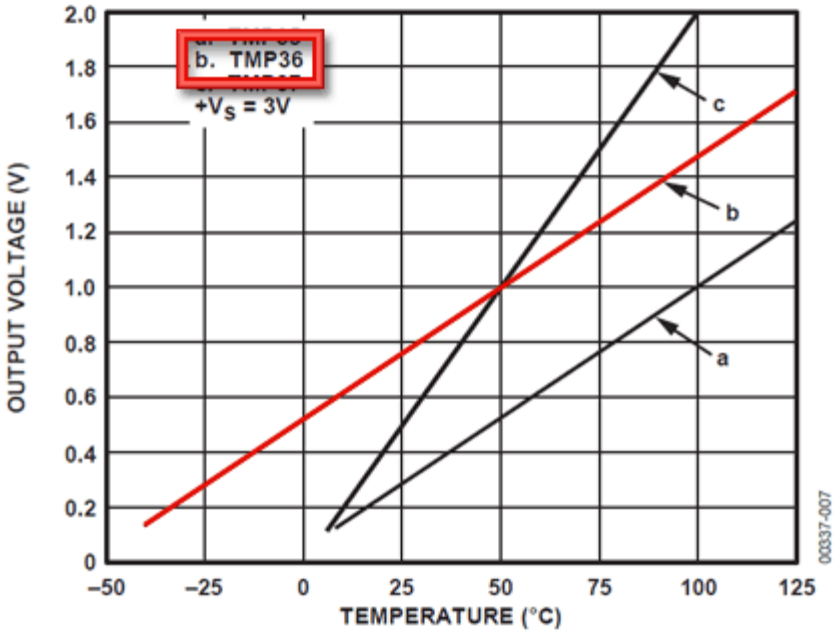


Fig: Connection diagram of the LM35DZ temperature sensor



To use the LM35 temperature sensor the left pin is connected to power and the middle pin to ground. Then the right pin will have an analog voltage that is directly proportional (linear) to the temperature. The analog voltage is independent of the power supply.

To convert the voltage to temperature, the following basic formula was used:

Temp in °C = (Vout in mV) / 10

In the above graph, line ‘a’ is used for the calculation of the temperature.

Testing the sensors

Connect a 2.7-5.5V power supply (2-4 AA batteries work fantastic) so that ground is connected to pin 2 (middle pin), and power is connected to pin 1 (left pin)

Then connect your multimeter in DC voltage mode to ground and the remaining pin 3 (right). The voltage at room temperature will be around 0.25V.

These sensors have little chips in them and while they're not that delicate, they do need to be handled properly. We have to be careful of static electricity when handling them and make sure the power supply is connected up correctly and is between 2.7 and 5.5V DC.

They come in a "TO-92" package which means the chip is housed in a plastic hemi-cylinder with three legs. The legs can be bent easily to allow the sensor to be plugged into a breadboard. We can also solder to the pins to connect long wires.

Reading the analog temperature data

To read the temperature value from the sensor we have to plug the output pin directly into an Analog (ADC) input. No matter what supply we use, the analog voltage reading will range from about 0V (ground) to about 1.75V.

Here we are using a 5V Arduino, and connecting the sensor directly into an Analog pin, we can use the following formula to turn the 10-bit analog reading into a temperature:

Voltage at pin in milliVolts = (*reading from ADC*) * (5000/1024)
This formula converts the number 0-1023 from the ADC into 0-5000mV (= 5V)
Then, to convert millivolts into temperature, use this formula:

Centigrade temperature = (analog voltage in mV) / 10

LM35 Precision Centigrade Temperature Sensors

General Description

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±1⁄4°C at room temperature and ±3⁄4°C over a full −55 to +150°C temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 μA from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a −55° to +150°C temperature range.The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package.

Features

1. Calibrated directly in ° Celsius (Centigrade)
2. Linear + 10.0 mV/°C scale factor
3. 0.5°C accuracy guarantee able (at +25°C)
4. Rated for full −55° to +150°C range
5. Suitable for remote applications
6. Low cost due to wafer-level trimming
7. Operates from 2.7 to 5.5 volts
8. Less than 60 μA current drain
9. Low self-heating, 0.08°C in still air
10. Nonlinearity only ±1⁄4°C typical

Strain Gage

Firdous Saleheen, Amrita Sahu and Tim Boger

A strain gage system capable of measuring strain was designed, tested and implemented. A circuitry for calculation of strain has been designed, simulated and implemented on breadboard. The circuit setup is then integrated with Arduino board for data acquisition. To test the performance an experiment, plastic balls of different elasticity and fruits of different hardness were used. The experimental result showed that the system can measure strain. From the measured strain can be calculated which is one parameter to calculate Young's modulus or hardness.

Strain is the amount of deformation of a body due to an applied force (1). More specifically, strain (e) is defined as the fractional change in length, as shown in Figure 1.

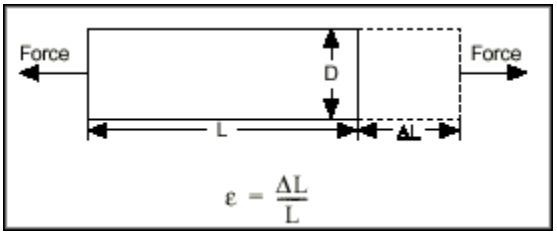


Figure 1. Definition of Strain

Strain can be positive (tensile) or negative (compressive). Although dimensionless, strain is sometimes expressed in units such as in./in. or mm/mm. In practice, the magnitude of measured strain is very small. Therefore, strain is often expressed as microstrain (me), which is e x 10^{-6}.

When a bar is strained with a uniaxial force, as in Figure 1, a phenomenon known as Poisson Strain causes the girth of the bar, D, to contract in the transverse, or perpendicular, direction. The magnitude of this transverse contraction is a material property indicated by its Poisson's Ratio. The Poisson's Ratio n of a material is defined as the negative ratio of the strain in the transverse direction (perpendicular to the force) to the strain in the axial direction (parallel to the force), or $n = e_T/e$. Poisson's Ratio for steel, for example, ranges from 0.25 to 0.3.

While there are several methods of measuring strain, the most common is with a strain gage, a device whose electrical resistance varies in proportion to the amount of strain in the device. The most widely used gage is the bonded metallic strain gage.

The metallic strain gage consists of a very fine wire or, more commonly, metallic foil arranged in a grid pattern. The grid pattern maximizes the amount of metallic wire or foil subject to strain in the parallel direction (Figure 2). The cross-sectional area of the grid is minimized to reduce the effect of shear strain and Poisson Strain. The grid is bonded to a thin backing, called the carrier, which is attached directly to

the test specimen. Therefore, the strain experienced by the test specimen is transferred directly to the strain gage, which responds with a linear change in electrical resistance. Strain gages are available commercially with nominal resistance values from 30 to 3,000 Ω, with 120, 350, and 1,000 Ω being the most common values.

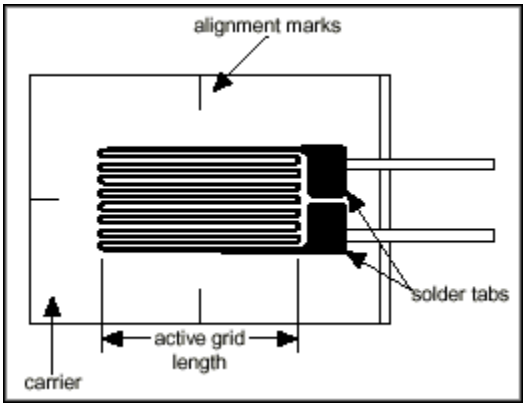


Figure 2. Bonded Metallic Strain Gage

It is very important that the strain gage be properly mounted onto the test specimen so that the strain is accurately transferred from the test specimen, through the adhesive and strain gage backing, to the foil itself.

A fundamental parameter of the strain gage is its sensitivity to strain, expressed quantitatively as the gage factor (GF). Gage factor is defined as the ratio of fractional change in electrical resistance to the fractional change in length (strain):

$$GF = \frac{\Delta R/R}{\Delta L/L} = \frac{\Delta R/R}{\varepsilon}$$

The gage factor for metallic strain gages is typically around 2.

In practice, strain measurements rarely involve quantities larger than a few millistrain (e x 10^{-3}). Therefore, to measure the strain requires accurate measurement of very small changes in resistance. For example, suppose a test specimen undergoes a strain of 500 me. A strain gage with a gage factor of 2 will exhibit a change in electrical resistance of only 2 (500 x 10^{-6}) = 0.1%. For a 120 Ω gage, this is a change of only 0.12 Ω.

To measure such small changes in resistance, strain gages are almost always used in a bridge configuration with a voltage excitation source. The general Wheatstone bridge, illustrated in Figure 3, consists of four resistive arms with an excitation voltage, V_{EX}, that is applied across the bridge.

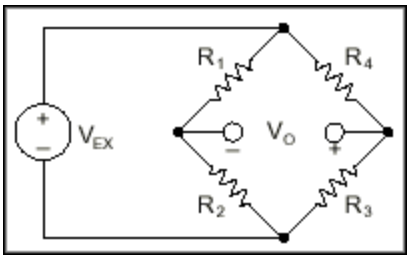


Figure 3. Wheatstone Bridge

The output voltage of the bridge, V_O, is equal to:

$$V_O = \left[\frac{R_3}{R_3 + R_4} - \frac{R_2}{R_1 + R_2}\right] \bullet V_{EX}$$

From this equation, it is apparent that when $R_1/R_2 = R_4/R_3$, the voltage output V_O is zero. Under these conditions, the bridge is said to be balanced. Any change in resistance in any arm of the bridge results in a nonzero output voltage.

Therefore, if you replace R_4 in Figure 3 with an active strain gage, any changes in the strain gage resistance will unbalance the bridge and produce a nonzero output voltage. If the nominal resistance of the strain gage is designated as R_G, then the strain-induced change in resistance, DR, can be expressed as $DR = R_G \cdot GF \cdot e$, from the previously defined Gage Factor equation. Assuming that $R_1 = R_2$ and $R_3 = R_G$, the bridge equation above can be rewritten to express V_O/V_{EX} as a function of strain (see Figure 4). Note the presence of the $1/(1+GF \cdot e/2)$ term that indicates the nonlinearity of the quarter-bridge output with respect to strain.

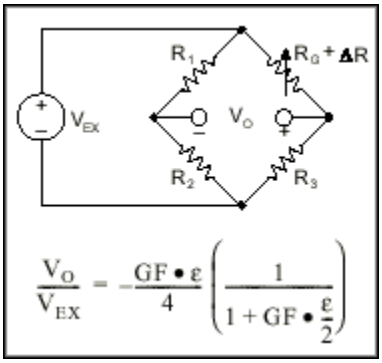


Figure 4. Quarter-Bridge Circuit

Ideally, you would like the resistance of the strain gage to change only in response to applied strain. However, strain gage material, as well as the specimen material to which the gage is applied, also responds to changes in temperature. Strain gage manufacturers attempt to minimize sensitivity to

temperature by processing the gage material to compensate for the thermal expansion of the specimen material for which the gage is intended. While compensated gages reduce the thermal sensitivity, they do not totally remove it.

By using two strain gages in the bridge, you can further minimize the effect of temperature. For example, Figure 5 illustrates a strain gage configuration where one gage is active (R_G + DR) and a second gage is placed transverse to the applied strain. Therefore, the strain has little effect on the second gage, called the dummy gage. However, any changes in temperature affect both gages in the same way. Because the temperature changes are identical in the two gages, the ratio of their resistance does not change, the voltage V_O does not change, and the effects of the temperature change are minimized. NOTE: In the Wheatstone bridge configuration, the active gage and the dummy gage should be on the same vertical leg of the bridge.

System Design and Strain Sensing Principle

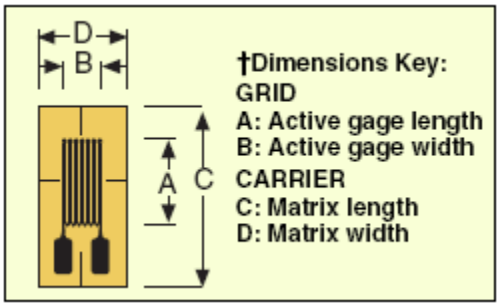
System Design

The system components are:

5. Force Sensing Resistor
6. Measurement Circuit
7. Arduino Board
8. Matlab

Strain Gage

SGT-2C/350-TY11 has been used as strain gage. The nominal value is 350 ohms. The dimension is given below:



A = 1.5mm, B = 4.6mm, C = 6.4mm and D = 4.6mm.

Circuit for Strain Measurement

Design

A sample circuit is given below whose example is followed for implementation. The example for the configuration of the circuit is given in Appendix. In this circuit the full bridge configuration has been used whereas in implementation stage quarter bridge configuration is followed. Some resistors have been changed due to availability constraints keeping equivalent resistor values. In order to facilitate remote

sensing, current excitation is used. The OP177 serves the bridge current to 10mA around a reference voltage of 1.235V. The strain gauge produces an output of ~0.5mV/1000me. The signal is amplified by the AD620 instrumentation amplifier which is configured for a gain of 100. Full-scale strain voltage may be set by adjusting the 100W gain potentiometer such that, for a strain of –3500me, the output reads –3.500V; and for a strain of +5000me, the output registers a +5.000V. The measurement may then be digitized with an ADC which has a 10V fullscale input range. The 0.1µF capacitor across the AD620 input pins serves as an EMI/RFI filter in conjunction with the bridge resistance of 1kW. The corner frequency of the filter is approximately 1.6k Hz.

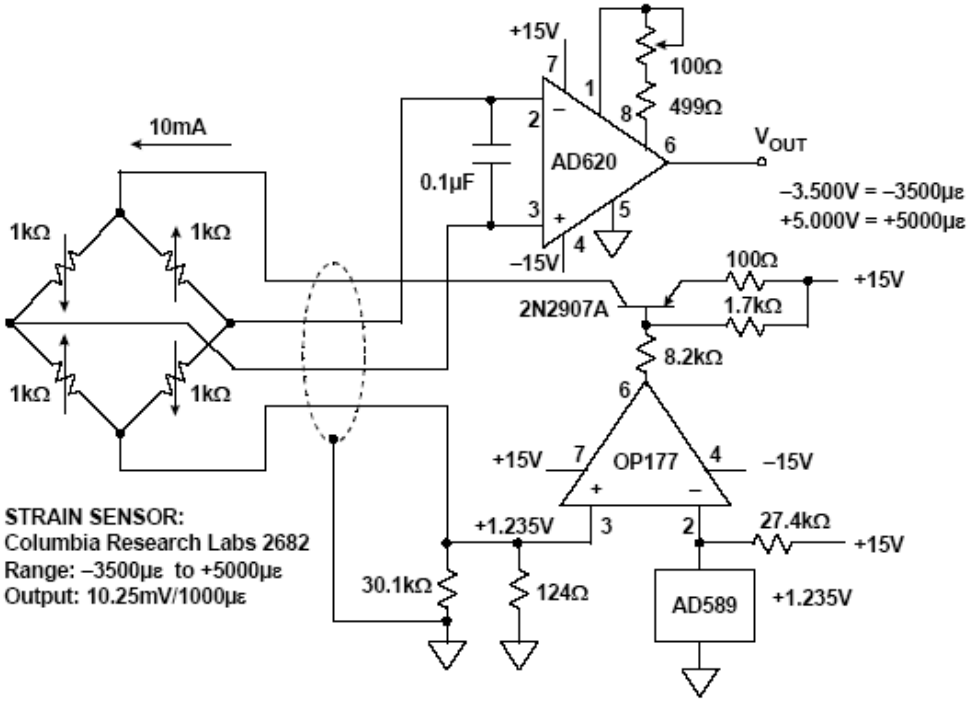


Figure 2: Sample Circuit for Signal Conditioning

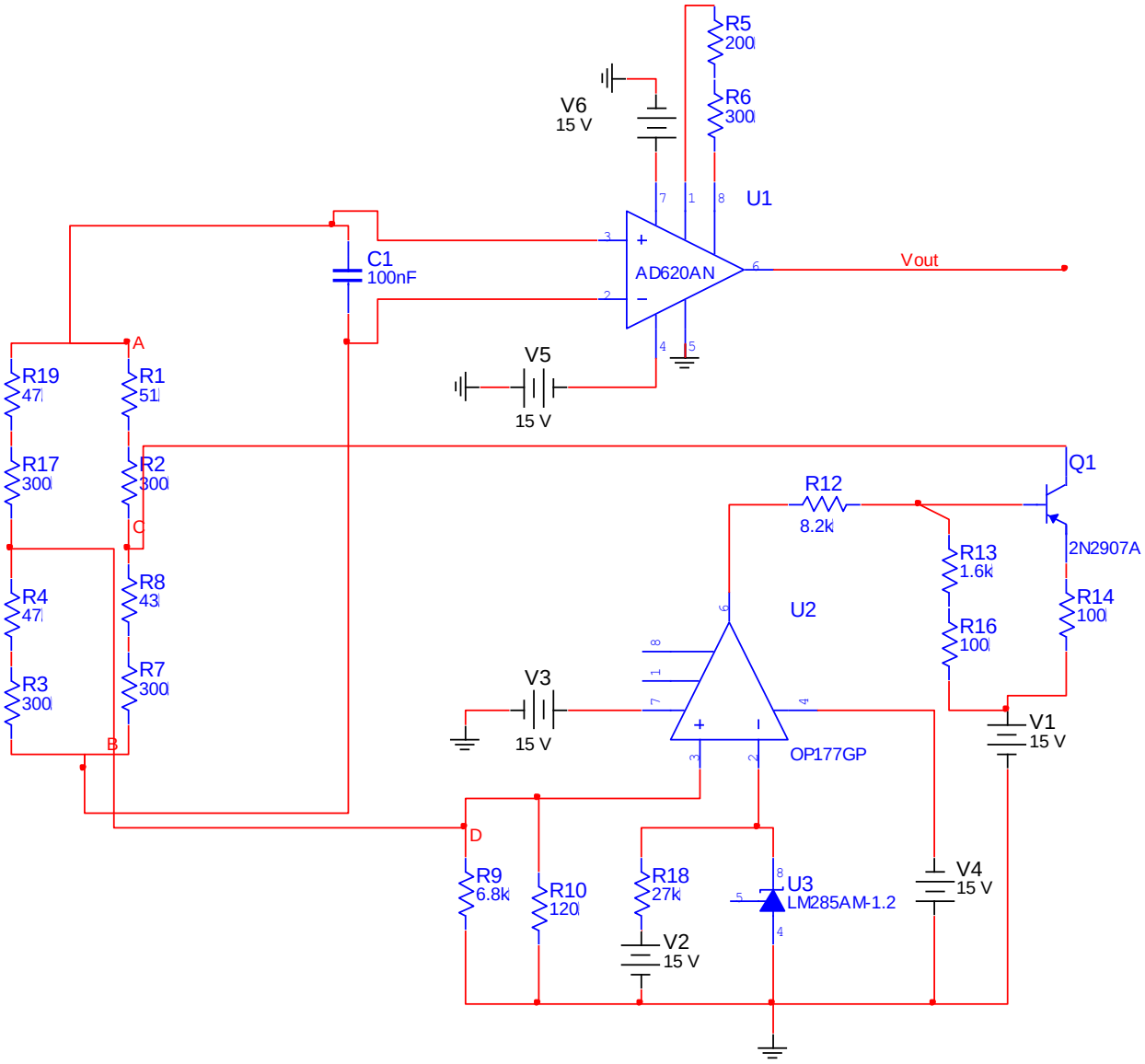


Figure 3: Strain Gage Wheatstone bridge and Signal Conditioning Circuit

Implementation

The circuit has been implemented in a breadboard.

Voltage and Strain Relationship

For calculation of strain from Vout, this relationship has been used:

strain = 4*v_strain/(2*100*(1200-2*v_strain))

Integration with Arduino Board and Data Acquisition

Integration with Arduino Board

The following materials are required and have been used for integration of strain gage with Arduino Board:

6. Matlab 2010b
7. Arduino Uno board (http://arduino.cc/)
8. Arduino IDE
9. Matlab support package for arduino known as ARDUINOIO
10. Strain gage with signal conditioning circuit

Steps for integration

1. See the Force Sensor section for the steps.

Data Acquisition Steps:

4. Assuming the above instructions are followed properly, following code should be run in matlab.
5. The code reads analogpin(2) of Arduino board and converts the serial data into voltage and then in strain.
6. The code also graphs the real time discrete data for voltage and strain.
    a. Matlab Code is given below

```
% Author :Firdous Saleheen
% date 11/14/2011
% read serial data from strain gage and plot against datapoints
clear all;
close all;
clc;
s1 = arduino('COM4');                    %define arduino object
%s1.BaudRate=9600;                        %define baud rate
%%
%initialize data acquisition parameters
clear data_strain;
numberofData = 100;
data_strain = zeros(1,numberofData);
v_strain = zeros(1,numberofData);
strain = zeros(1,numberofData);
i = 1;

for i= 1:numberofData                %acquisition of 100 points
  % read analog data from sensor
   data_strain(i) = s1.analogRead(2);
   v_strain(i) = data_strain(i)*(10000/1024);
 figure(1);
subplot(2,1,1);
title('Voltage and Strain data plot');
 stem(i,v_strain(i));
xlabel('Number of datapoints');
   ylabel('Voltage (mV)');
   drawnow;
   grid on;
hold on;
% strain voltage relation
   subplot(2,1,2)
```

```
    strain(i) = 4*v_strain/(2*100*(1200-2*v_strain));
    stem(i,strain(i),'r');
    xlabel('Number of datapoints');
    ylabel('Strain (strain)');
    drawnow;
    grid on;
    hold on;
end


delete(s1)
```

Application Software

See details in Force Sensor section

Experimental Setup and Procedures

Experimental setup

2. For the experimental setup the following materials are required:
    - Matlab 2010b
    - Arduino Uno board
    - Arduino IDE
    - Matlab support package for arduino known as ARDUINOIO
    - Strain gage with circuit
    - Avocado of different hardness
    - Plastic balls of different hardness
    - Robot arm

Procedure for ball hardness experiment

12. Matlab, Arduino IDE and ARDUINOIO package should be installed into the workstation.
13. Arduino Board is for controlling robot arm and reading serial data from sensor.
14. Through the program in IDE the robot arm should reach the target (avocado) and touch it with the mounting plastic board with strain gage installed in its claw.
15. The matlab code should be run now which will read the serial port and return the real time voltage and strain data.

Procedure for Fruit ripeness check experiment

1. The strain will be calculated using the above method and then will be used in application software.

Experimental Results

Ball hardness check experiment has returned the graph of figure 3.

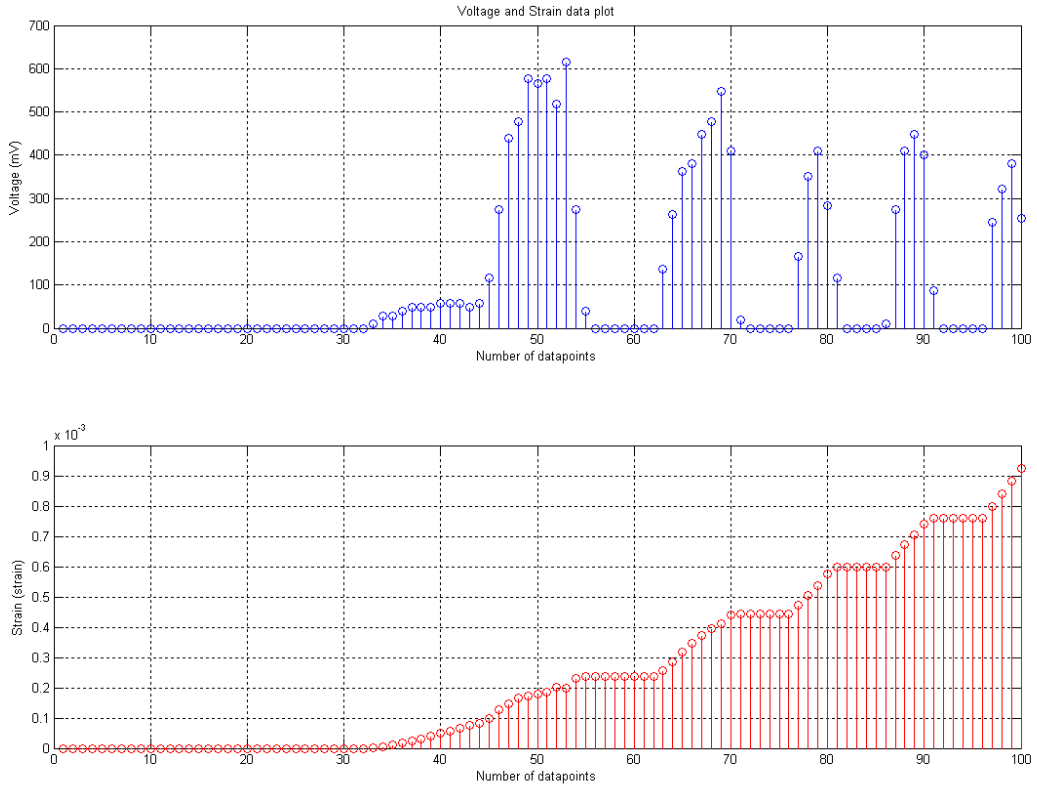


Figure 4: ball experiment sensor data

Discussion

Conclusion

Acknowledgement

This project is sponsored by Dr. Chang-Hee Won. The authors would also like to thank Dr. Won for his valuable advice and Matt Bosack for suggestions

Appendices

Appendix-A Program Code

Arduino Code: Basic Robotic Arm Functionality

```
#include <AFMotor.h>
AF_DCMotor motor1(1, MOTOR12_1KHZ);
AF_DCMotor motor2(2, MOTOR12_1KHZ);
AF_DCMotor motor3(3, MOTOR12_1KHZ);
AF_DCMotor motor4(4, MOTOR12_1KHZ);
const int pingPin = 2;
int POT = 0;
int count = 0;
int x = 0;

void setup() {
  Serial.begin(9600);
  motor1.setSpeed(200);
  motor2.setSpeed(200);
  motor3.setSpeed(200);
  motor4.setSpeed(200);
}

void loop()
{
//************************************************************
//PING SENSOR
  long duration, inches, cm;
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);

  pinMode(pingPin, INPUT);
  duration = pulseIn(pingPin, HIGH);

  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);
//************************************************************
//MOTORS
  POT = analogRead(1);

      if(cm <= 8)
      {
        POT = analogRead(1);
        delay(3000);
        while(POT >= 0 && POT < 12)
        {
        motor2.run(FORWARD);
        POT = analogRead(1);
```

```
        }
        motor2.run(RELEASE);


        delay(2000);
        motor1.run(FORWARD);
        delay(1000);
        motor1.run(RELEASE);
        delay(2000);
        motor1.run(BACKWARD);
        delay(900);
        motor1.run(RELEASE);

         POT = analogRead(1);
        while(POT >1 && POT < 22)
        {
        motor2.run(BACKWARD);
        POT = analogRead(1);
        }
        motor2.run(RELEASE);
        delay(5000);

      }
      else
      {
      motor1.run(RELEASE);
      }

  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm, ");
  Serial.print(POT);
  Serial.print("units,  ");
  Serial.println();

  delay(100);

}

long microsecondsToInches(long microseconds)
{
  // According to Parallax's datasheet for the PING))), there are
  // 73.746 microseconds per inch (i.e. sound travels at 1130 feet per
  // second).  This gives the distance travelled by the ping, outbound
  // and return, so we divide by 2 to get the distance of the
obstacle.
  // See: http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.3.pdf
  return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds)
```

```
{
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the
  // object we take half of the distance travelled.
  return microseconds / 29 / 2;
}
```

Appendix B – Curve Fitting

Relationship between force and voltage

General model Power2:

f(x) = a*x^b+c

Coefficients (with 95% confidence bounds):

a = -2617 (-3811, -1424)

b = -0.1054 (-0.153, -0.05782)

c = 6835 (5659, 8011)

x = force

f(x) = voltage

Goodness of fit: SSE: 2023, R-square: 0.9986, Adjusted R-square: 0.9982, RMSE: 17

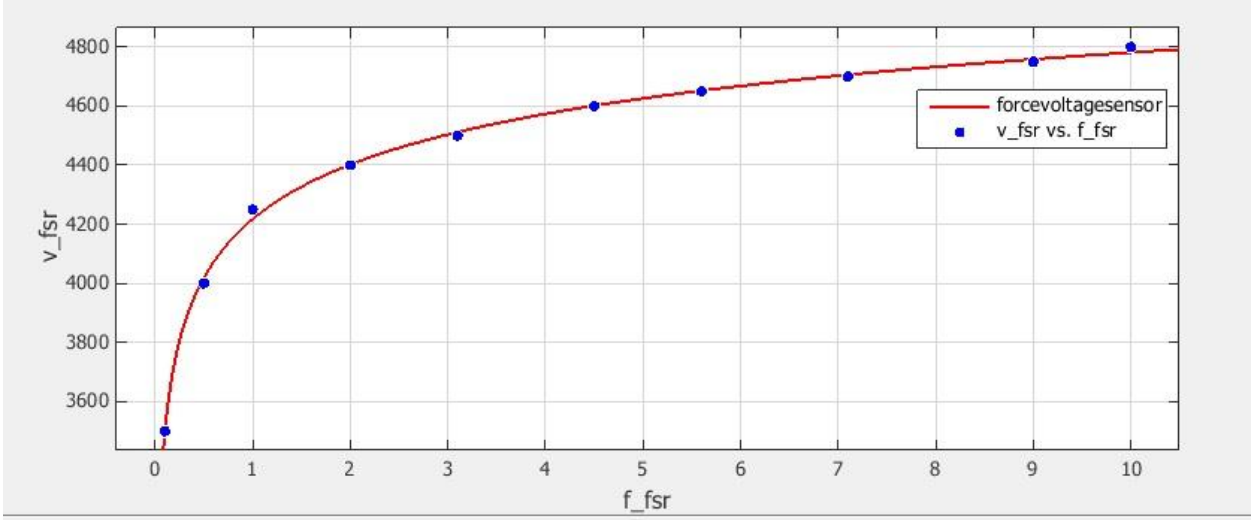


Figure 21:Output Voltage(v_fsr) vs Force(f_fsr) curve fitting

From the above relation it can be written,

force = exp(log((voltage-6835)/(-2617))/(-0.1054))

Matlab Code

```
%Author Firdous Saleheen
% Date 11/14/11
% data from interlink manufacturer of fsr
clear all
global f_fsr;
global v_fsr;
% x axis data force in newton (N)
f_fsr = [0.1 0.5 1 2 3.1 4.5 5.6 7.1 9 10];
% y axis data voltage in mV
v_fsr = [3500 4000 4250 4400 4500 4600 4650 4700 4750 4800];


function [fitresult, gof] = forcevoltagecreateFit(f_fsr, v_fsr)
%CREATEFIT(F_FSR,V_FSR)
%  Create a fit.
%
%  Data for 'forcevoltagesensor' fit:
%      X Input : f_fsr
%      Y Output: v_fsr
%  Output:
%      fitresult : a fit object representing the fit.
%      gof : structure with goodness-of fit info.
%


%% Fit: 'forcevoltagesensor'.

[xData, yData] = prepareCurveData( f_fsr, v_fsr );
%%
% Set up fittype and options.
ft = fittype( 'power2' );
opts = fitoptions( ft );
opts.Display = 'Off';
opts.Lower = [-Inf -Inf -Inf];
opts.StartPoint = [4129.32491634478 0.0738716357522151 -5.05373756995814];
opts.Upper = [Inf Inf Inf];

% Fit model to data.
[fitresult, gof] = fit( xData, yData, ft, opts );

% Plot fit with data.
figure( 'Name', 'forcevoltagesensor' );
h = plot( fitresult, xData, yData );
legend( h, 'v_fsr vs. f_fsr', 'forcevoltagesensor', 'Location', 'NorthEast' );
% Label axes
xlabel( 'f_fsr' );
ylabel( 'v_fsr' );
grid on
```

Appendix C – Quarter bridge type configuration for strain gage

This section provides information for the quarter-bridge strain-gauge configuration type II. The quarter-bridge type II measures either axial or bending strain.

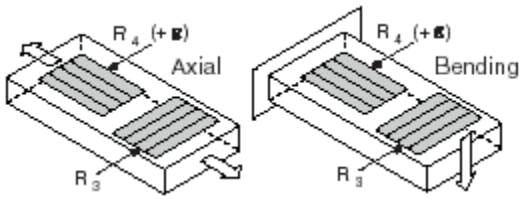


Figure 1-4. Quarter-Bridge Type II Measuring Axial and Bending Strain

A quarter-bridge type II has the following characteristics:

• One active strain-gauge element and one passive, temperature-sensing quarter-bridge element (dummy gauge). The active element is mounted in the direction of axial or bending strain. The dummy gauge is mounted in close thermal contact with the strain specimen but not bonded to the specimen, and is usually mounted transverse (perpendicular) to the principle axis of strain.

• This configuration is often confused with the half-bridge type I configuration, with the difference being that in the half-bridge type I configuration the R_3 element is active and bonded to the strain specimen to measure the effect of Poisson's ratio.

• Completion resistors provide half bridge completion.

• Compensates for temperature.

• Sensitivity at 1000 me is ~ 0.5 mV_{out}/ V_{EX} input.

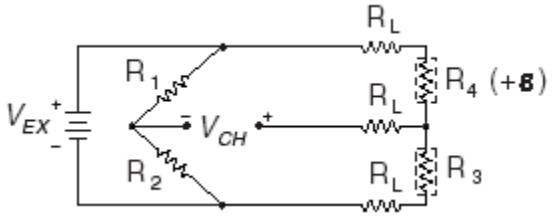


Figure 1-5. Quarter-Bridge Type II Circuit Diagram

The following symbols apply to the circuit diagram and equations:

• R_1 and R_2 are a half-bridge completion resistors.

• R_3 is the quarter-bridge temperature-sensing element (dummy gauge).

• R_4 is the active strain-gauge element measuring tensile strain (+e).

To convert voltage readings to strain units use the following equation