Bachelor Thesis

School of Engineering

June 2012

Distance Laboratory-
Measurement of Signal Propagation in an Iron Bar

Yunfei Wang, Xi Zhang

School of Engineering
Blekinge Institute of Technology
SE-37179 KARLSKRONA, SWEDEN
Phone: +46 45585000

Contact Information
Authors:

Yunfei Wang
E-mail: wangyunfeisweden@gmail.com

Xi Zhang
E-mail: xizhangsw@gmail.com

Supervisor:
Anders Hultgren
Email: anders.hultgren@bth.se
Section/Unit: ING School of Engineering

Blekinge Institute of Technology
School of Engineering
Phone: +46 455 3850 00
Fax: +46 455 38 50 57
E-mail: info@bth.se
Webpage: www.bth.se/eng

ABSTRACT

In the laboratory it should be possible to perform an impulse response experiment of the iron bar. A mechanical system for generating the impulse input the end of the iron bar should be developed.

At the one end of the Iron bar there are strain gauge sensors and measurement amplifiers mounted. It should be possible to connect one channel from an oscilloscope to the output form the measurement amplifiers.

Contents

Chapter 1

Introduction

This report introduces a distance laboratory including measurement of the pressure wave propagation in an iron bar.

The experiment, measurement of wave propagation in an iron bar will be arranged to be managed as distance laboratory. Anyone can perform the experiment via internet from anywhere by using PC connected to the distance laboratories IP address at the Blekinge Institute of Technology, BTH, Sweden. Via a virtual front panel the whole experiment can be controlled and possibly several students will enjoy this experiment.

1.1 Distance Lab

Distance Lab is a research initiative for digital media technology and design innovation, focused on addressing the many problems and opportunities found in rural and remote areas of the world. In addition to conducting academic research, Distance Lab works with briefs from industry and governmental partners, providing advice, generating ideas and building prototypes that can lead to new products and services.

Laboratory exercises in electrical engineering courses can be performed remotely using real equipment. A number of user-defined experiments have been conducted over Internet at Blekinge Institute of Technology (BTH), Sweden. The experiments have been carried out in different locations using the same experimental hardware located in a small closed laboratory at BTH.

1.2 The Mechanical system (Iron Bar)

The Iron bar that will be used in the experiment is about 0.6 meter long and has a diameter of about 0.05meter. The material is steel. The iron bar is part of the equipment used in Atlas Copco machines such as breakers as figure 2.2.2 shown.

In order to describe the iron bar system we derive a mathematical model of the mechanical system. A mechanical system can be modeled by use of the variables force and velocity and by using of three parameters mass, spring constant, and dissipation.

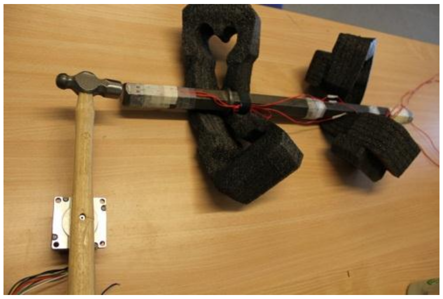


Figure.1 Mechanical system

Iron is elastic to some extent. The elasticity is given by the material property Bulk modulus. The elasticity is increasing with increasing length of the bar, and is decreasing by increasing cross section area of the bar. A spring constant in the axial direction of a bar can be calculated. The mass for a bar also can be calculated is the density for iron. A mechanism whith a hammer connected to a motor is used for hitting the iron bar end and introduce the pressure wave. This mechanism will be described in chapter 4 and chapter 5.

1.3 Strain gauge sensor

A strain gauge sensor is used to measure press wave inside the iron bar. The strain gauge shape is the one that shown as right figure and it can transform length of mechanical structures to resistance changing. It is glued on a mechanical structure, and if the mechanical structure is changed in length, the glued resistance is also changed in length. According to the formula 1.1, the resistance *R* changes dependent on the length *L*. The glued resistance is connected to a Wheatstone bridge that measures the resistance with high accuracy. The measurement signal is proportional to the change in length of the mechanical structure.

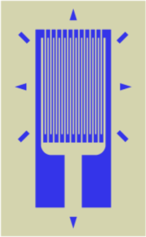


Figure.2 Strain gauge sensor

$$\mathrm{R} = \rho \cdot \frac{L}{A} \qquad (1.1)$$

When a pressure applied on it but do not exceed its limit (do not break it), the mechanical structure will become longer or narrower. Electrical resistance will increases or decrease respectively. See figure.3 and formula 1.1.

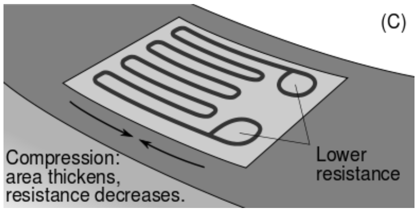


Figure.3 Visualization of the Working Concept

This is one way which from the measured electrical resistance of the strain gauge, the amount of applied stress can be inferred.

1.4 Internet interface

Due to the project dependent on internet control, the users have to set the remote connecting between the remote PC and the local PC.

There are a lot of ways to perform it. If you using WIN 7 system, it is easy to create a password in the local PC and make it remoteable and then open the *mstsc* in the remotely PC and setting the local IP address and password.

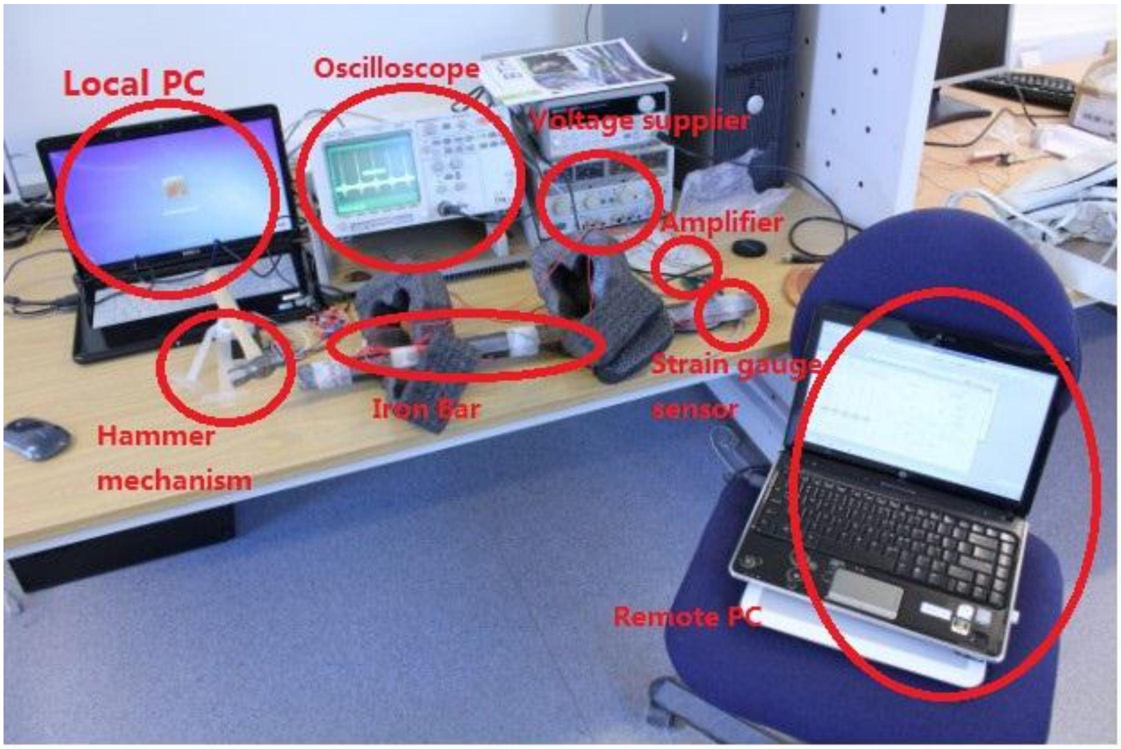


Figure.4 Remote control system

The picture.4 shows the final remote control system. It include one local PC, one oscilloscope, one iron bar, one strain sensor glued on iron bar, one sensor amplifier, one voltage supplier for the amplifier, one hammer mechanism and one remotely PC.

1.5 Measurement & Modeling results

Below figure.5 and figure.6 are the measurement result which got from oscilloscope by the force generated on the iron bar and the modeling result of this experiment. We can see there is a lot of reflection that because of the pressure wave traveling in the iron bar caused multi reflection. Figure.6 shows the simulation result and figure.5 shows the signal from measurement. The time in between the peaks in both figures are the travelling time spent on return trip on iron bar.

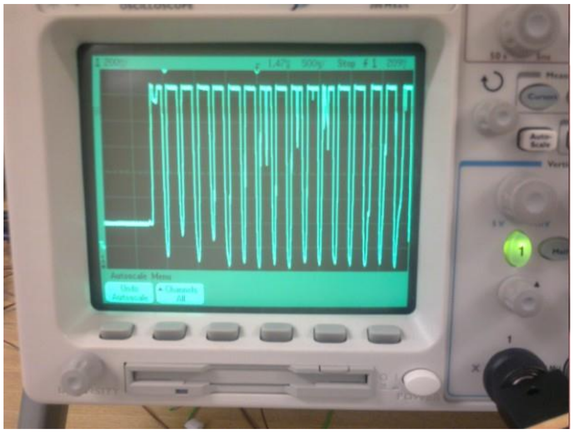


Figure.5 Measurement result

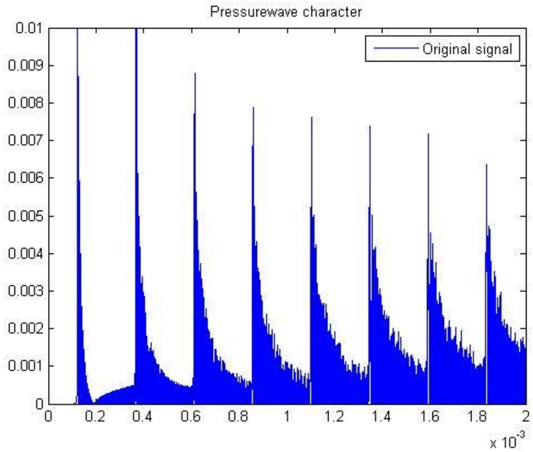


Figure.6 Simulation result

Where the left hand figure.5, measurement result, we got is coming from a 200KHz filter, we can see that the time delay between each two peek is 250us (0.25×10^{-3}s). The right hand figure.6, simulation result, shows the original pressure wave. We also can see that the time delay between each two peeks is around $0.25 \times 10^{-3}s$. This time delay is the time that press wave takes to travel around the iron bar. So the velocity can be calculated by the length of iron bar and half time delay see formula 1.2. Where *d* is length of the iron bar 0.6*m* and the *T* is half time delay. Comparing with the theoretical velocity 5000 *m/s* as calculated by formula 3.13 in section 3.3.1, we can say that we have a deviation that can be accepted.

$$\mathrm{v} = \frac{\mathrm{d}}{\mathrm{T}} = \frac{0.6m}{\frac{1}{2} \times 0.25 \times 10^{-3}s} = 4800m/s \tag{1.2}$$

In this report, there are seven chapter included. The chapter 2 introduces some background of distance lab and pressure wave in iron bars. The chapter 3 introduces how to model the pressure wave and some description of sensor and filter selection. The chapter 4 introduces the hardware- stepper motor and chapter 5 introduces the hardware- stepper motor controller also the PC controlling interface of the stepper motor controller. The chapter 6 describes the PC controlling interface of the oscilloscope used in this project. The chapter 7 describes the measure of pressure wave also describe some hardware connection of the strain gauge sensor with related amplifier.

Chapter 2
Background

2.1 Distance Lab

Distance learning has been promoted across the entire education sector due to the increasing number of people that educate themselves after their working hours or as part of their professional development. Furthermore, universities and high schools already use the Internet extensively for communication with their students. This, combined with the recent developments with regards to the internet and information technology, has seen the need for internet-based teaching grow rapidly.

In engineering education of Universities, most of experiments require instruments or some expensive equipment to be performed. Some equipments are very sensitive with temperature, and some of them are too heavy to move, so the distance laboratories are used by Blekinge Institute of Technology. The BTH provides one that convenient to control and with exactly same experience as traditional experiment. As so far there are some examples like Vienna University's distance laboratories, see [18], Kumamoto University's distance real laboratory system, see [19]. The BTH also provides traditional lab sessions in a remote laboratory for circuit analysis see [15].

In our distance lab, the local PC is connected with a motor and an oscilloscope and the oscilloscope is connected with a strain gauge sensor amplifier. See [2],[3]. Users can use their PC control local PC over internet to active the motor and hammer to knock the iron bar and the sensor will generate signals that can be captured by the oscilloscope also can be seen by user PC.

2.2 Pressure waves in Iron Bars

When applying an impulse forces at one terminal of a bar in axial direction, a pressure wave is generated and propagated along the bar. It is possible to model wave propagation in a bar by meaning of modeling the Iron bar as l lumped model consisted of a large set of rigid masses connected by springs such as figure.7 illustrated. The reference about this iron bar modeling is at reference [4].

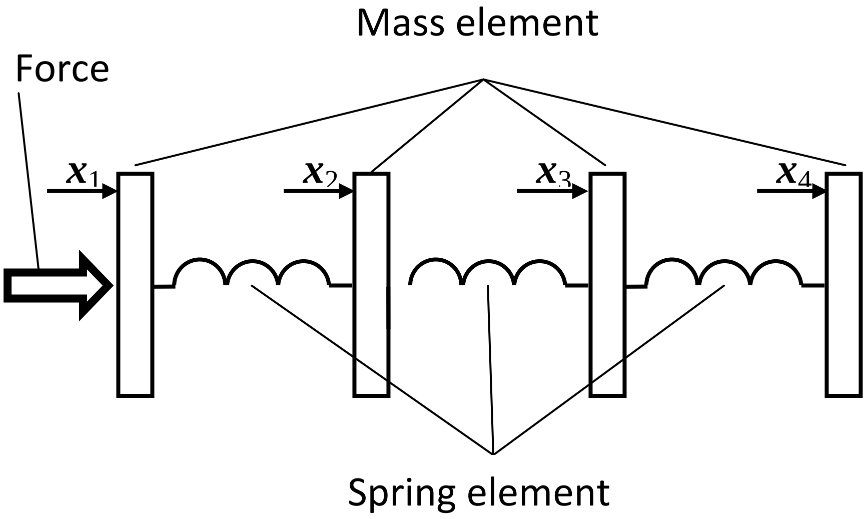


Figure.7 Model of a lumped bar

Figure.7 shows a model of a lumped bar. The bar is sectioned in four mass parts with intermediate springs. The coordinate x_i means the positions of the each mass element.

The formula of modeling this spring mechanical system is as formula 2.1.

$$f = K(x_1 - x_2) \tag{2.1}$$

When a force generated at one terminal of the iron bar, this force will transmit itself element by element until the force is consumed. During this process, the element will be extended or be compressed. And it also will make the sensor glued on the surface of iron bar extend or compress. So the resistance of sensor is changing when we knock the iron bar and for this reason, we can measure the pressure wave by using strain gauge sensor. Section 3.1and 3.2 describe how to model this mechanical system and section 3.3 introduces the sensor in detail which will be used.

Figure.8 Breaker

There are a lot of industrial applications apply this system such as breakers as you can see from figure.8. See [20].

Chapter 3

Modeling and Simulation for pressure wave in an Iron bar

This chapter is going to introduce how to model and simulate the pressure waves propagated in an Iron bar.

3.1 Modeling and simulation of system

We will model a steel bar from Altas Copco having the length of about 600mm. The Iron bar is modeled as a lumped model with a high number of elements. The higher number of elements, the better accuracy is achieved. A very high number of elements will give a long simulation time. The number of elements is chosen according to the resolution of the used sensor for measuring the pressure wave in the iron bar. The sensor and the measurement amplifier are chosen in order to get a low cost system. The modeling and simulation is performed with 500 elements, corresponding to a lumped iron bar model having about one mm long elements. Determining the mass and spring constant for each of elements and derive an equation system for the bar. The equation system will be given as a system matrix with a diagonal band structure.

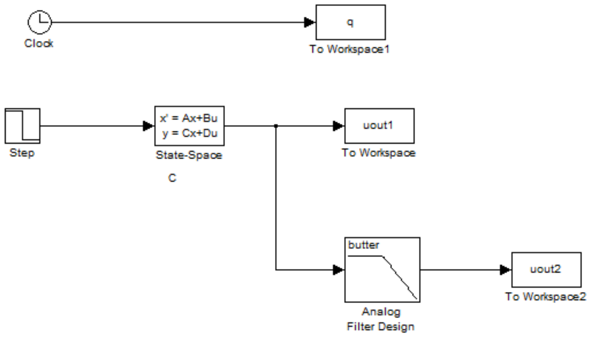


Figure.9 Simulation of the system

A model is designed in MATLAB and Simulink. The Simulink model can use ABC-model. That is shown in figure.9.

3.2 ABC model of signal propagation

Building of model of previous four elements is shown as formula 3.1.

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \ddot{x}_1 \\ \ddot{x}_1 \\ \ddot{x}_1 \\ \ddot{x}_1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{-k}{m} & \frac{k}{m} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{k}{m} & \frac{-2k}{m} & \frac{-k}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{k}{m} & \frac{-2k}{m} & \frac{k}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{k}{m} & \frac{-2k}{m} & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{m} \\ 0 \\ 0 \\ 0 \end{pmatrix} \mathrm{F} \qquad (3.1)$$

Where the parameters' meaning in above model are described as Table 1:

Table 1 Description of parameters

Parameter	Description
x_j	position of each element
$\dot{x}_j$	velocity of the pressure wave propagation into Iron bar
$\ddot{x}_j$	Accelerated velocity of the pressure wave propagation into Iron bar
k	Spring constant
m	Mass of each element
F	Input signal (the force knocking at the end of iron bar)

The spring constant follows formula 3.2:

$$K = E \cdot A/L \qquad (3.2)$$

Where *K* is spring constant, *E* is young's modulus 1.9×10^{11}pa, *A* is area of transverse surface $1.96 \times 10^{-3} m^2$ and *L* is length of iron bar 0.6m.

The initial equations of designing whole system are written as below:

$$\ddot{x}_1 = \frac{1}{m}[F - K(x_1 - x_2)] \quad (when\, j = 1) \qquad (3.3)$$

$$\ddot{x}_j = \frac{1}{m}\left[K\left(x_{j-1} - x_j\right) - K(x_j - x_{j+1})\right] \quad (when\, j > 1) \qquad (3.4)$$

The ABC-differential equation system is given as

$$\begin{cases} \dot{x} = \mathrm{Ax} + \mathrm{Bu} \\ \mathrm{y} = \mathrm{Cx} \end{cases} \qquad \begin{array}{r} (3.5) \\ (3.6) \end{array}$$

For this case we measure the velocity at the end of bar: $\dot{x}_4$. Where the *u* is the input signal, the *y* is the output we want to figure out.

$$\mathrm{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{-k}{M} & \frac{k}{M} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{k}{M} & \frac{-2k}{M} & \frac{-k}{M} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{k}{M} & \frac{-2k}{M} & \frac{k}{M} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{k}{M} & \frac{-2k}{M} & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.7}$$

$$\mathrm{B} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{M} \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.8}$$

$$\mathrm{C} = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 1) \tag{3.9}$$

For getting more accurate simulate, we use 500 spring elements instead of 4 elements and it can be generated via MATLAB. The reason of why we choose 500 elements is described in section 3.1.

In the simulation as figure.10, apply a short quick force pulse with high amplitude onto the terminal of the bar in our simulated model.

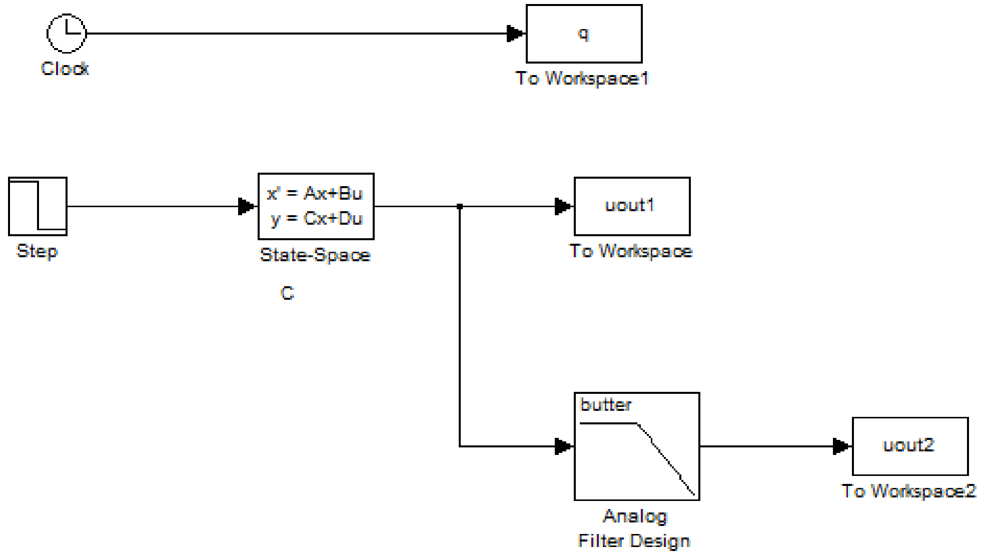


Figure.10 Simulation of signal propagation in Iron bar system

Where the *'uout1'* is the original signal (pressure wave) and *'uout2'* is the signal went through adding filter. During the propagation wave traveling, the noise will happen so we need to add a filter to let the propagation wave more clearly to analysis. The setting parameters are given by the Table 2:

Table 2 PARAMETERS SETTING

Step	Initial Value	1000 units (give an amplitude)
	Step Time	11×10^{-6} s (To simulate a very short time Force that can be assume as a pulse signal)
State-Space	A,B,C,D parameters	Give certain variables which can be set matrix in with the same variables in m-file for these parameters.
Filter	Low-pass Filter	Order=8
	Pass-band Edge Frequency	Setting bandwidth performed by angular frequency

After setting related parameter in the simulation blocks and connected with ABC model, we can run this MATLAB programming and get the pressure wave as figure.11. This figure shows a simulation which included the modeled pressure wave propagated in an Iron bar and one added filter to describe the measurement system. The most important cost related property of the measurement system is the bandwidth. In order to model the limited bandwidth a low-pass filter is added in the simulation.

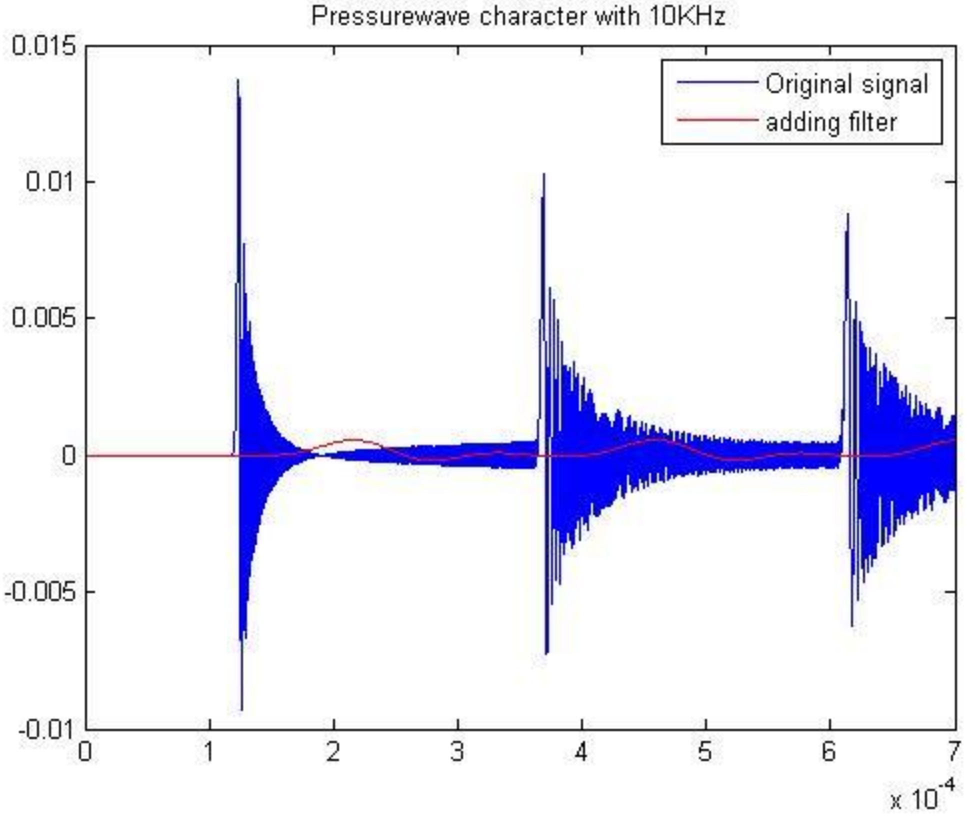


Figure.11 Signal comparison with 10KHz filter

In figure.11, the amplitude does not have real meaning but for the real propagation wave it will mean the voltage, the blue pulses sequence represents the original pressure wave and red pulses sequence is the wave went through the addition 10KHz bandwidth filter. Here using 10KHz bandwidth filter just for taking an example to see how the original signal and the adding filter's signal look like. We use different measurement amplifier with different bandwidth in order to find a suitable bandwidth for the amplifier, using the iron bar element length of about one mm. We also can see, in the figure.11 there are a lot of reflections that because the pressure wave travelled in the iron bar over and over and the time between two top points is the travelling time that pressure wave travelled one round trip.

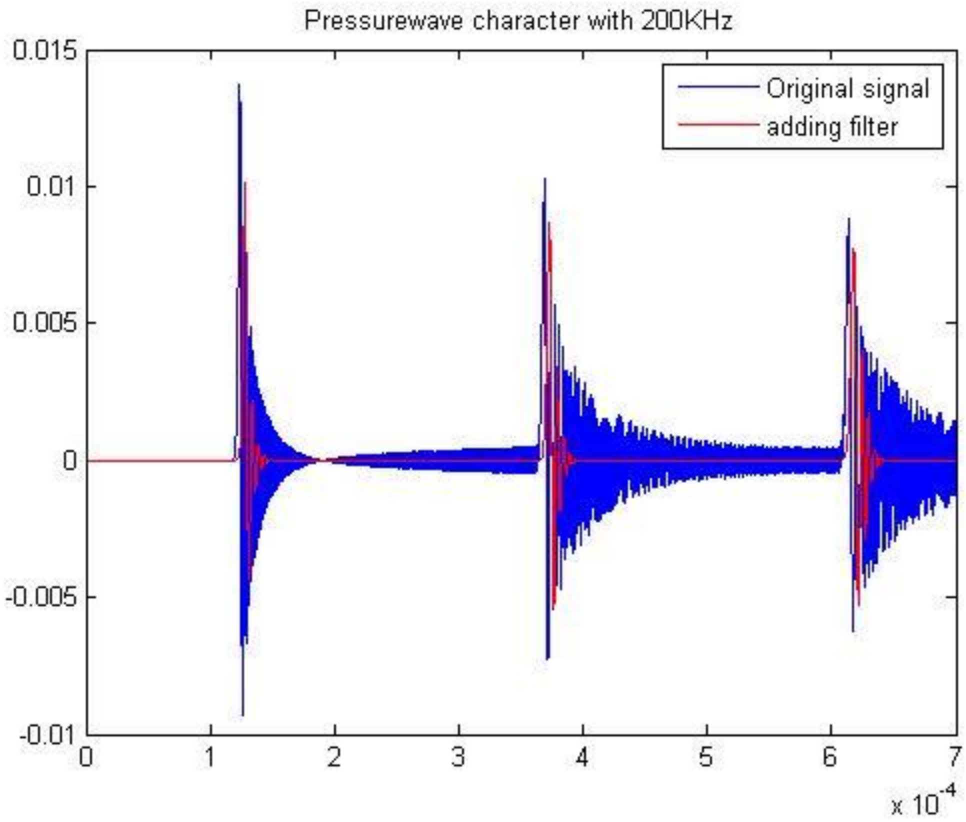


Figure.12 Signal comparison with 200KHz filter

As comparison the filter we have chosen in the real measurement has 200KHz bandwidth. In order to have a visual representation of that filter, we also show the simulation result with 200KHz filter as above figure.12 for comparing with 10KHz filter. The simply reason of why the 200KHz filter be used is that it has broad bandwidth and has low enough price and the detail analysis will be introduced in section 3.3.2.

3.3 Sensor and filter selection

In order to perform this experiment with physical equipment, we need choose a suitable strain gauge sensor and an amplifier with suitable physical filter.

3.3.1 Sensor

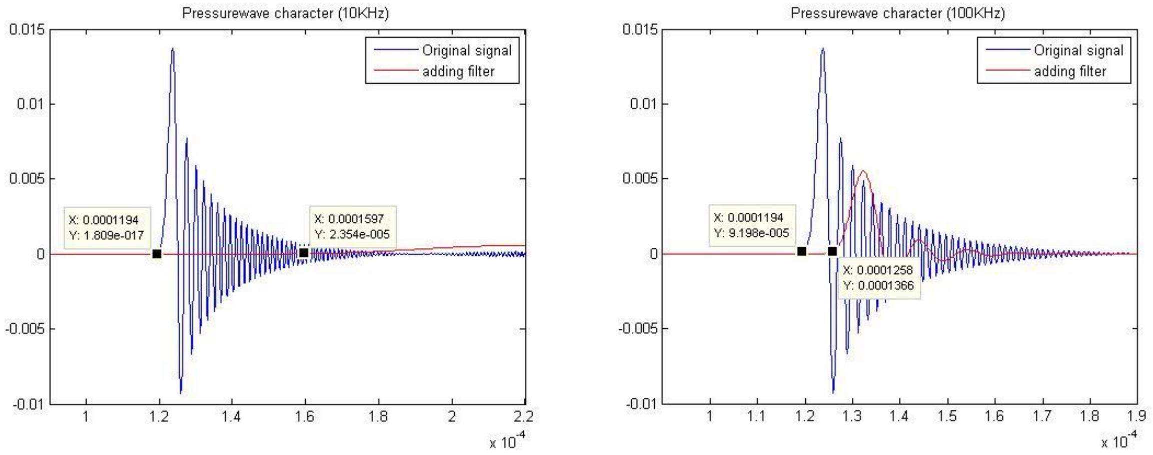
The strain gauge sensor has a certain length. The sensor measurement of the pressure wave is an average across the sensor length. In order to measure the propagation wave the sensor length has of course to be significantly shorter compared to the iron bar length. The shorter it is chosen the more accurate measurement signal will be generated. A very short sensor will possibly give a too low measurement signal amplitude. To match the sensor choice the amplifier has to have a suitable bandwidth. A short strain gauge sensor asks for a high amplifier bandwidth.

In order to keep the cost not too high, we have chosen a standard strain gauge sensor, and not very high bandwidth in the amplifier. The used strain gauge sensor has the length 1 mm which can be found in the data sheet of KFG-1-350-C1-11. A sensor length of one mm is consistent with having about 500 sections in the simulation of the 600mm long iron bar.

3.3.2 Amplifier

Different bandwidth will generate different averaging of the signals.

Comparison with different bandwidth 10KHz, 100KHz, 200KHz, 1MHz, is shown as figure.13 and from these comparison, it is easy to see that during bandwidth increasing, and the signal went through filter became as same as simulated result signal. It can be judged that a 200KHz bandwidth filter seems consistent with choosing about one mm resolution in the sensor choice and in the simulation element length. So the strain gauge amplifier with 200KHz is used in this project and in the chapter 7, it describes how to use this amplifier.



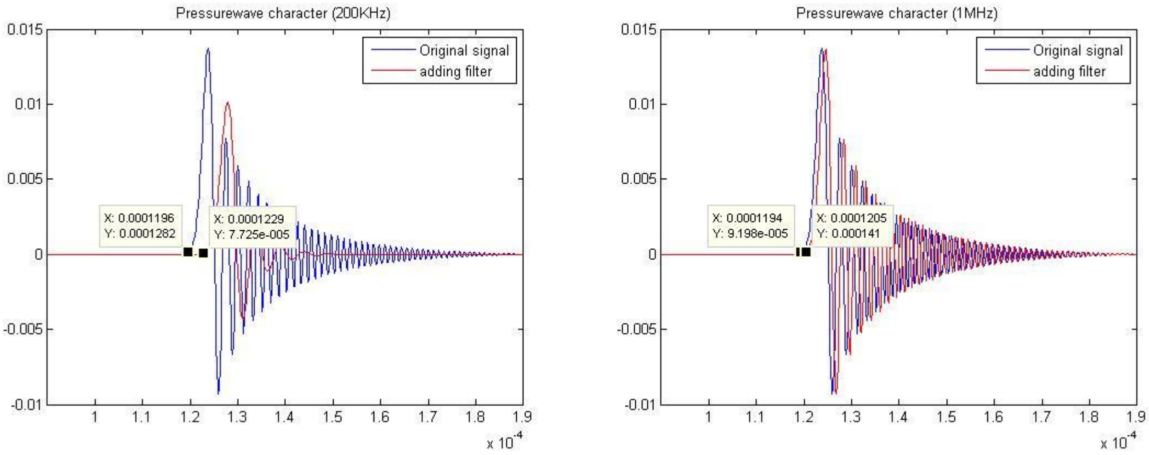


Figure.13 Filter comparison with different bandwidth

Give the sensor we try to find a suitable bandwidth in the amplifier on base of how the sensor affect the pressure wave.

The velocity of pressure wave propagation, v, in a solid iron bar is given by formula 3.10. The velocity is dependent on density, and elastic properties of the media, see [11].

$$\mathrm{v} = \sqrt{\frac{elastic\ property}{inertial\ property}} = \sqrt{\frac{B}{\rho}} \qquad (3.10)$$

Where **B= Bulk modulus, ρ= density.**

The Bulk modulus of Iron is 200 GPa and density is 7.874g $\cdot\, cm^{-3}$

$$7.874\ \mathrm{g}/cm^{3} = 7874\ \mathrm{Kg}/m^{3} \qquad (3.11)$$

$$200\mathrm{GPa} = 200 \times 10^{9}\ \mathrm{N}/m^{2} \qquad (3.12)$$

The velocity of propagation as below:

$$\mathrm{v} = \sqrt{\frac{200 \times 10^{9}\ N/m^{2}}{7874\ Kg/m^{3}}} = 5039.846\ \mathrm{m}/s \qquad (3.13)$$

Our strain gauge sensor on the Iron bar has 1mm length. The wave will travel across the sensor during T seconds, where T is given by:

$$\mathrm{T} = \frac{\mathrm{d}}{\mathrm{v}} = \frac{0.001m}{5039.846\ m/s} = 0.198\ \ \mu\mathrm{s} \qquad (3.14)$$

In order to have a good measurement from the chosen sensors we need choose a suitable bandwidth in the amplifier.

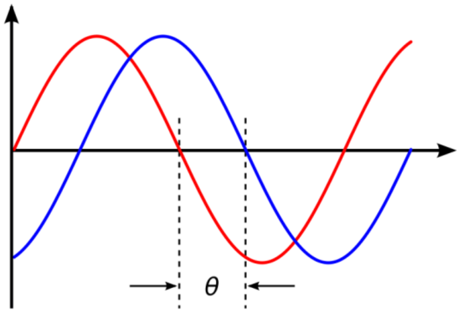


Figure.14 Phase shift

We introduce a concept that is phase lag. As we can see on the above figure.14, the red line preform original signal and blue line preform the signal passed low-pass filter.

The assumed phase lag in the measurement sensor implied that we have not use for a very high bandwidth in the measurement amplifier. Let's assume that we choose a filter bandwidth that not further will affect the measurement signal. We then state that the delay should be visible as a visible phase shift at the bandwidth. Below bandwidth, f, calculations are shown for phase shifts between $\frac{\pi}{3}$ to $\frac{\pi}{30}$

$$\frac{\pi}{30} \leq \ \omega\mathrm{T} \ \leq \frac{\pi}{3} \tag{3.15}$$

$$\frac{\pi}{30} \leq \ 2\,\pi\,f\ \mathrm{T} \ \leq \frac{\pi}{3} \tag{3.16}$$

$$\frac{1}{60\mathrm{T}} \leq \ f \ \leq \ \frac{1}{6\mathrm{T}} \tag{3.17}$$

$$\frac{1}{60 \times 0.198\ \mu\mathrm{s}} \leq \ f \ \leq \ \frac{1}{6 \times 0.198\ \mu\mathrm{s}} \tag{3.18}$$

$$\frac{1}{60 \times 0.22 \times 10^{-6} s} \leq \ f \ \leq \ \frac{1}{6 \times 0.22 \times 10^{-6} s} \tag{3.19}$$

$$0.084 \times 10^{6}\mathrm{Hz} \ \leq \ f \ \leq \ 0.84 \times 10^{6} \tag{3.20}$$

$$84\mathrm{KHz} \leq \ f \ \leq 840\mathrm{KHz} \tag{3.21}$$

So we can say that if we select one first order filter had frequency between 84KHz and 840KHz, the delay time of 0.198 µs is enough visible in the output.

In this chapter we discussed how to select a suitable filter for matching the original signal. In order to get a good filter that do not change the strain gauge signal too much, but has a suitable price, we chose different bandwidth to simulate pressure wave to see how the signal changing.

The model we used in this chapter was separated the entire iron bar by 500 elements. Since the length of iron bar is 0.6m, the each element is 1.2mm length. Compared with sensor we used had 1 mm, the length of each iron bar element is short enough. So this model can be used to investigate which suitable bandwidth is. By other words, if the filter does not affect the simulated signal it should not affect the measured signal too much.

Chapter 4

Stepper Motor

Before select one motor, we need to know how much torque is enough for moving hammer. For this testing, we put the iron bar vertical and let an iron ball free fall from the iron bar with certain high until the ball hit the end of iron bar. After several testing, we find the pressure wave can be performed very well when the iron ball free fall from iron bar with 15cm high. In this case, the torque is calculated by formula 4.1.

$$\tau = mgh = 0.3kg \cdot 0.98N/kg \cdot 15cm = 4.41Ncm \tag{4.1}$$

Where the *m* is the weight of iron ball, *g* is acceleration of gravity and *h* is the height from iron bar. So the 4.41*Ncm* torque is enough. We choose this unipolar stepper motor which has 90N.cm holding torque. Stepper motor's advantages are very easy to control the speed, rotated direction and determine the shaft position.

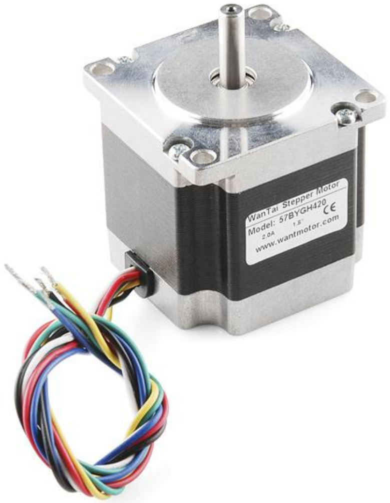


Figure.15 Stepper Motor

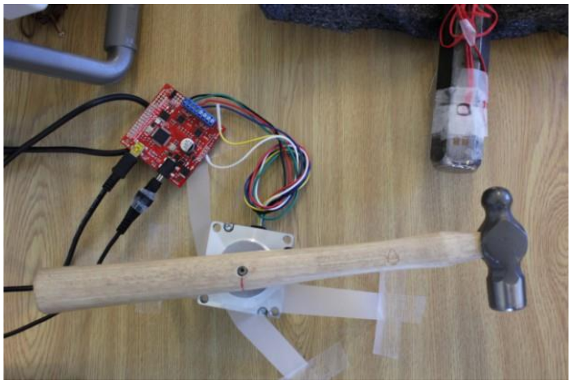


Figure.16 Hammer system

4.1 Types of stepper motor

Stepper Motors family be made up of a variety of sizes, and strengths. The basic types of stepper motor are Bipolar and Unipolar. The bipolar stepper usually has 4 wires. The unipolar stepper usually has 5, 6 or 8 wires. This chapter is going to discuss how unipolar stepper works and how to differ with bipolar stepper.

4.2 Stepper motor operation principle

Stepper motor consist of a permanent magnet rotating shaft, normally called the rotor, and electromagnets on the stationary position that surrounds the motor, called the stator.

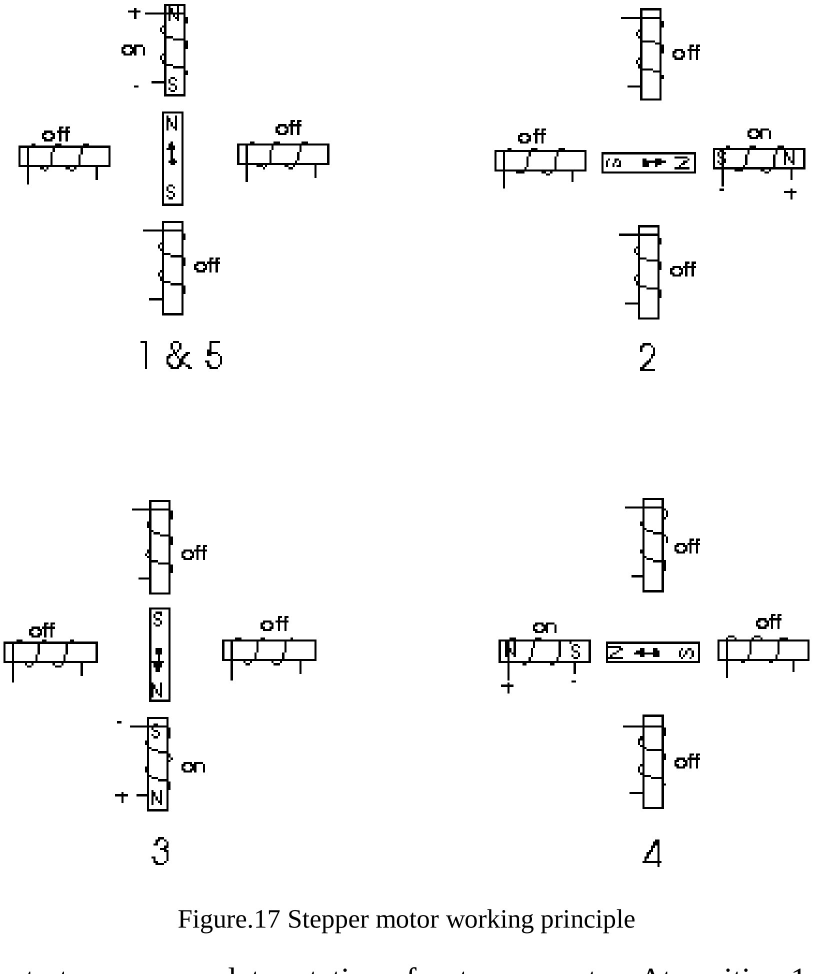


Figure.17 Stepper motor working principle

Figure.17 illustrates one complete rotation of a stepper motor. At position 1, we can move the rotor clockwise, the upper electromagnet is deactivated and the right electromagnet is activated, causing the rotor to move 90 degrees clockwise, aligning itself with the active magnet.

This process is repeated in the same manner at the south and west electromagnets until we once again reach the starting position.

4.3 Unipolar stepper motor

The Unipolar stepper motor has 2 coils, simple lengths of wound wires. The coils are identical and are not electrical connected. Each coil has a center tap which is the one wire coming out from the middle of coil' length as figure.18 shown.

Since a center tap is added between the two leads, uni-directional current flow in each ½ of winding. See figure.19 purple arrowhead

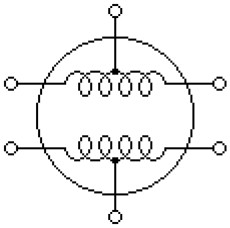


Figure.18 Unipolar stepper motor's 2 coils

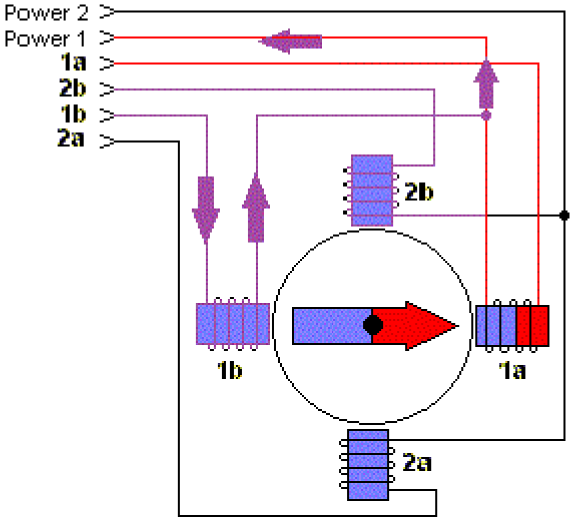


Figure.19 Conceptual Model of Unipolar stepper Motor

4.4 Bipolar stepper motor

The Bipolar stepper motor has 2 coils. The coils are identical and not electrically connected. It has a single winding per phase. The current in a winding needs to be reversed in order to reverse a magnetic pole, so the driving circuit is more complicated.

Each lead is taken separately, and Bi-directional current flow through entire winding at a time. See blue arrowhead in the Figure.21.

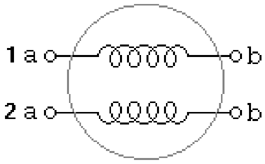


Figure.20 Bipolar stepper motor's 2 coils

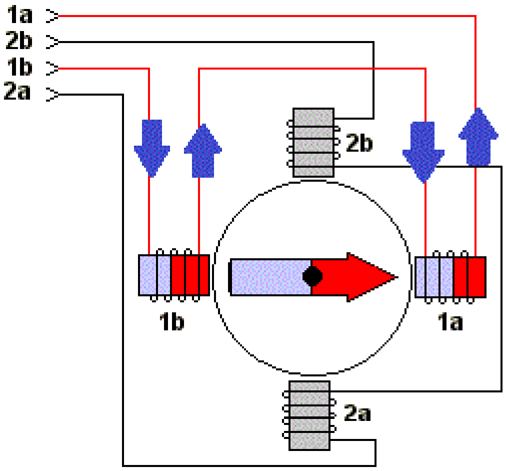


Figure.21 Conceptual Model of Bipolar Stepper Motor

4.5 Drive uni-polar stepper by bipolar driver

The certain stepper motor must need a matched driver, so unipolar stepper motor has to be driven by unipolar driver and bipolar stepper has to be driven by bipolar driver, respectively. In some case we have to use bipolar driver to control unipolar stepper. So the special connection is needed. For connecting unipolar stepper with bipolar driver, it need ignore the center tap and rest of them can connect as bipolar stepper's way. The detail connection is shown as figure.22.

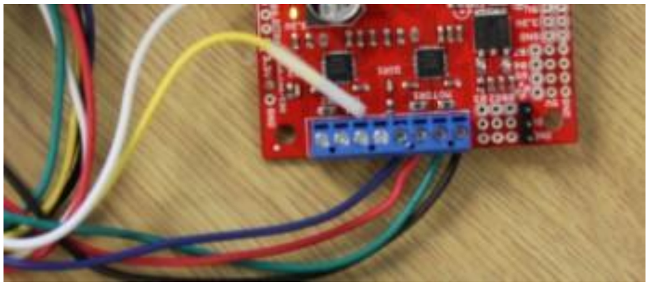


Figure.22 Connection of Uni-polar Stepper and Driver

Chapter 5

Stepper motor controller

Since we already choose one stepper motor, In order to control it in real time by a PC we need to choose one controller to control it. We just decide to use EiBotBoard (EBB) to complete this mission. This chapter also discuss how to connect EiBotBoard (EBB) stepper motor controller with stepper motor and how to programing it to achieve real time controlling.

5.1 EiBotBoard (EBB) Overview

This EiBotBoard was originally designed for the Egg-Bot project. It is a two stepper motor driver with USB microcontroller. It is developed based on UBW board and it supports all of commands of UBW.

The way of this board controls two stepper motors moving is sending command from a PC over a USB connection. This is accomplished by two micro-stepping chopper stepper motor drivers and a Microchip PIC microcontroller which supports USB connection. It will be known by a PC as a serial port when you plug this EBB board into your PC' USB port. That will let you can very easy to build your own application to send commends to EBB for controlling stepper motors.

Need to note that different board version have different features as some output changed or some requirements changed. In this project we use the EiBotBoard version 1.4 as figure.23 shown.

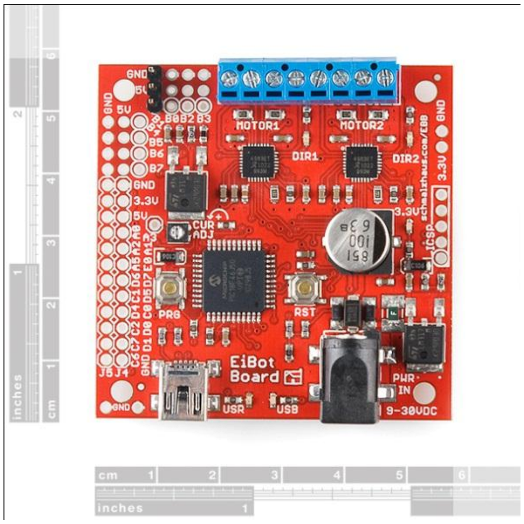


Figure.23 EiBotBoard

5.2 Hardware connection

The EiBotBoard consists of a PIC18F46J50 microcontroller and two Allegro A4983 stepper derivers. It works with some voltage regulators and USB connection hardware. And it takes 6-24V DC input with one barrel jack connector.

This directly powers the motor driver chips and then be dropped down to 5V to supply power to RC servos and down to 3.3V to power the microcontroller. You can set the maximum current allowed to motors from 46mA to 1.25A per phase by adjusting the current adjustment potentiometer which in the center of the EBB. More two push buttons allow for resetting the EBB and entering into bootloader mode (to update the firmware via USB) which will be described on section 5.4 updating EBB firmware. Three LEDs indicate the 3.3V power, stepper direction, and USB connection status. The screw terminals support for connecting with many types of stepper motor, but they are must either be 4-wire stepper motor or 6, 7, 8-wire motors. The reason is the two Allegro A4983s are bi-polar driver chips, so any stepper motors which purposed connecting with EBB are need wired in bi-polar mode.

Blueprint of EiBotBoard is shown as figure.24. In order to control ours uni-polar stepper motor, MOTRO 1or MOTOR 2 can be used for connecting with the motor as bi-polar model. Power supply and USB connector must be connected. Since the motor which we already choose need 2A /phase to get maximum torque, so when the motor is driven by EBB its maximum torque will be approximate 40N/cm.

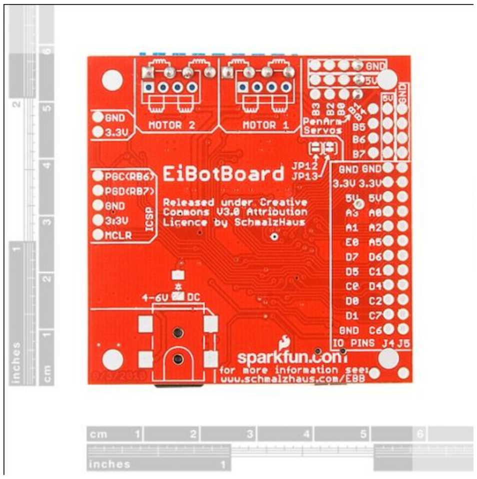


Figure.24 EiBotBoard back side

The figure.25 shows how the motor system connected.

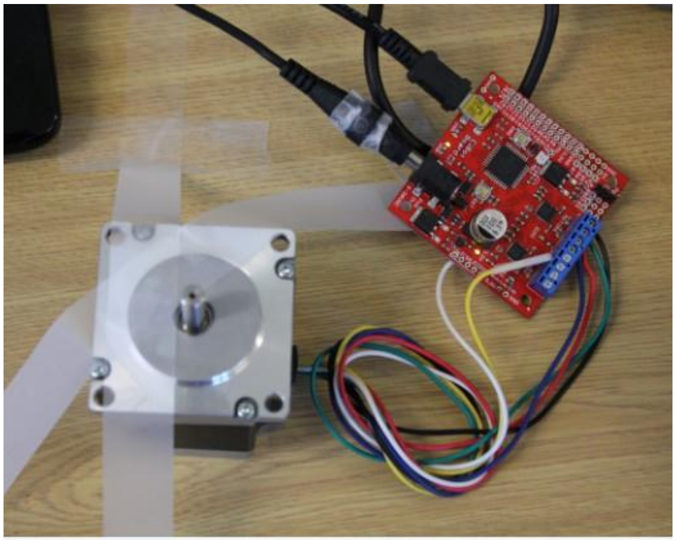


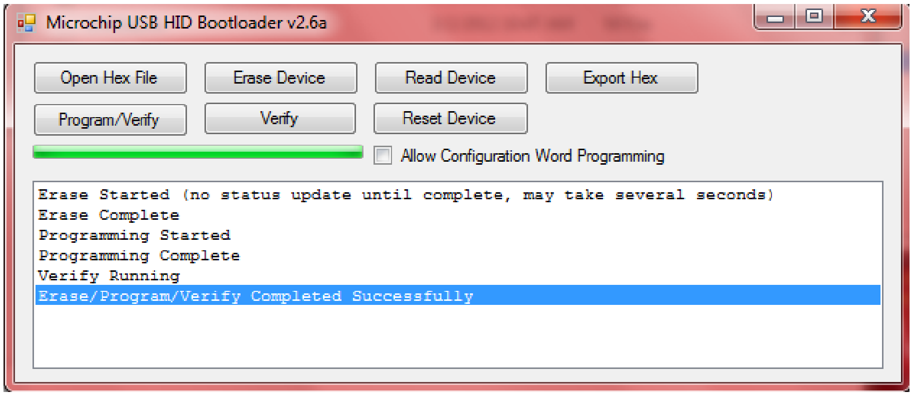
Figure.25 System connection of the motor

5.3 Software setting

When first time using this EBB in a PC, it need install a UBW driver to be known by the PC. After that it can be plugged into your computer via a USB cable. And this time it will show up as a serial port. When it is done installing (and your EiBotBoard' LED is slowing flashing), you can see this device's port number even changing the port number. The detail steps are shown in the Appendix II.

5.4 Updating EBB firmware

We use this EBB by Windows7 and need update firmware before using this board. In this procedure, the HIDBootLoader.exe, updating software is going to be used. You will find this software on the webpage of EBB. It also needs a USB cable, a power supply for EBB, a HEX file downed from website that you want to program into this EBB. When it is successful you will see the figure.26. The detailed steps are introduced in appendix III.



```
Erase Started (no status update until complete, may take several seconds)
Erase Complete
Programming Started
Programming Complete
Verify Running
Erase/Program/Verify Completed Successfully
```

Figure.26 EBB firmware updating

5.5 Used commands

This EiBotBoard' firmware is based on the UBW firmware. The same basic command processing framework is used, so the same types of commands are used, and the same types of errors are returned. There are list some important commands used in our project and some explanation.

The "V" Command (version testing)

- Format: "V"

- Return Packet: “EBBv 13_and_above EB Firmware Version 2.1.5”

The “EM” Command (enable motor)

- Format: “EM, <Enable 1>, <Enable 2>”
    - To enable a motor driver, set its <Enable> parameter to 1.
    - To disable a motor driver, set its <Enable>parameter to 0.
    - To set the microstep mode of BOTH motor driver (the same signals go to both drivers, so you can’t set them separately) use a value of 1,2,3,4 or 5 for <Enable 1>. When you use a value of 1,2,3,4 or 5 for <Enable 1>, the <Enable 2> parameter is not needed.
    - When setting microstep values with <Enable 1>:
        - 1 will enable both axis in 1/16st step mode (default on boot)
        - 2 will enable both axis in 1/8 st step mode
        - 3 will enable both axis in ¼ st step mode
        - 4 will enable both axis in ½ st step mode
        - 5 will enable both axis in full step mode
    - Note that any time an SM command is executed, both motor become ‘enabled’ before the move starts, thus it is almost never necessary to issue a “EM,1,1” command to enable both motors.
- Example : “EM,2” –this command will set both motor in 1/8st step mode.
- Return Packet: “OK”

The “SM” Command (Stepper motor move)

- Format: “SM,<duration>, <axis 1>, <axis 2>”
    - <duration> is a value from 1 to 65,535 and is in milliseconds. It represents the total length of time you want this move to take.
    - <axis 1> and <axis 2> are values from -32,767 to +32,767 and represent the number of steps for each motor to take <duration> milliseconds.
    - If both <axis 1> and <axis 2> are zero, then a delay of <duration> ms is executed. <axis 2> is an optional value, and if it is not included in the command, zero steps are assumed for axis 2.
- The maximum speed that the EBB can generate is 25,000 steps/s.
- Example : “SM,1000,250”
- Return Packet: “OK”

5.6 Drive with motor

Now we need connect the driver and motor and the connection way has been explained in the Charpter 4. So we just use that way to screw the motor on the EBB. Figure.27 shows the six wires out coming from motor. We just ignore two of middle wires when connected with EBB, since the motor is uni-polar and EBB is prepared for bi-polar motor.

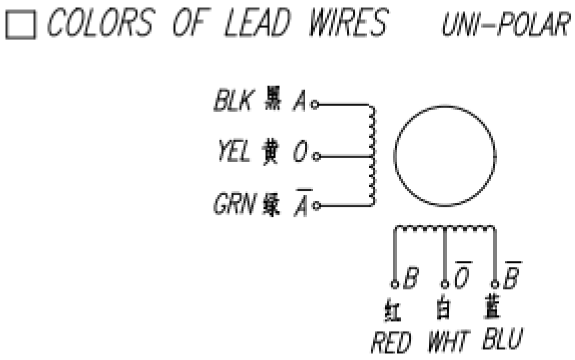


Figure.27 Wires Distribution

The Yellow one and White one in figure.27 are center wires will be ignored. The final connection is represented as figure.28.

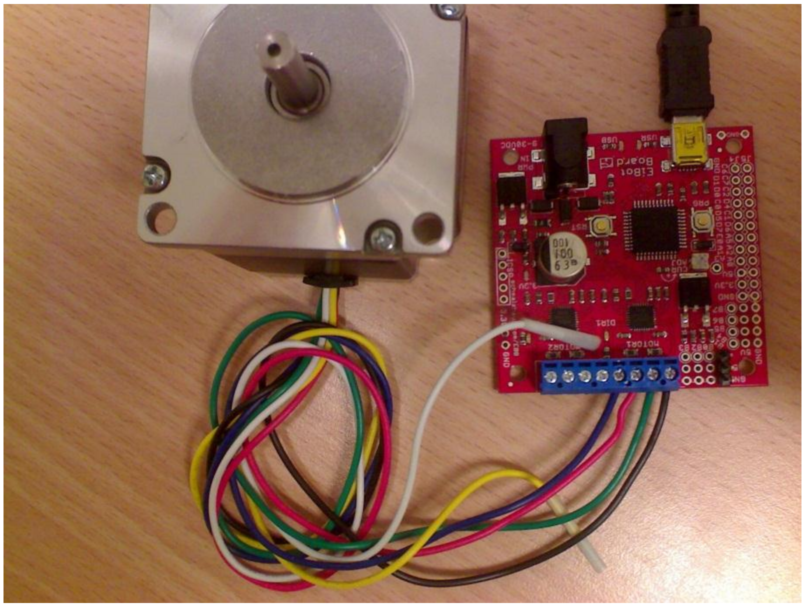


Figure.28 Connection of motor and driver

5.7 Generate force with hammer and motor

In this section the mechanical system will be combined completely. Connect a hammer with the motor to generate a force via PC controlling. But we need drill some holes for inserting into the motor firstly.

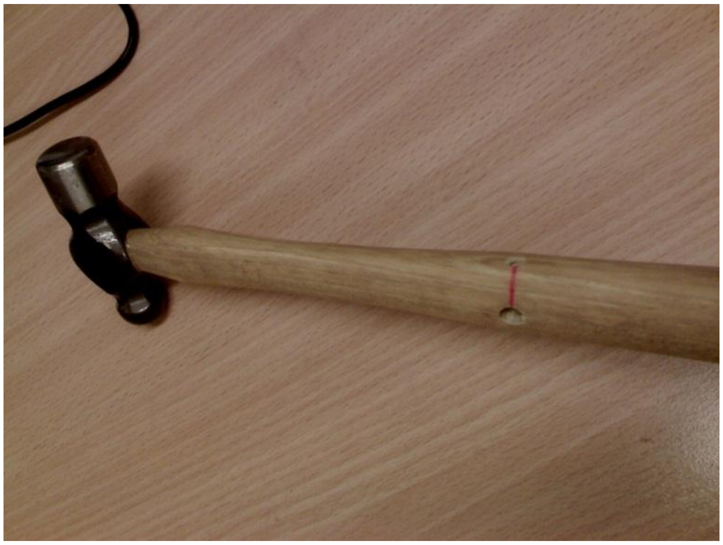


Figure.29 Hammer

There are two holes in the figure.29, the one is the main hole for inserting into motor and another is a screw hole which will be screwed a nut for fixing the motor with the hammer.

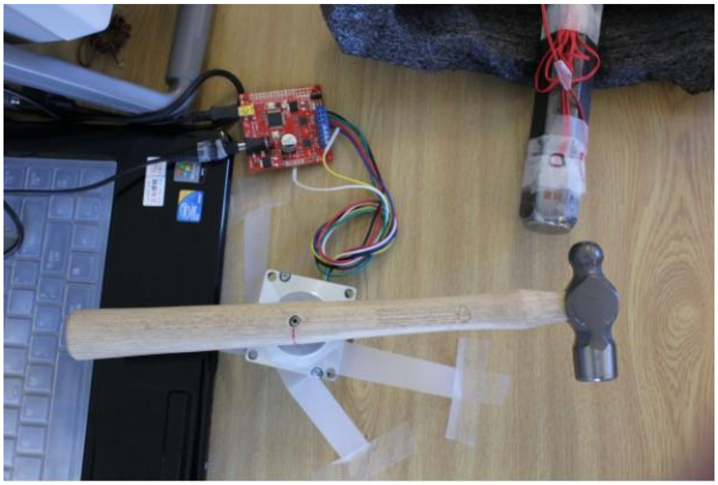
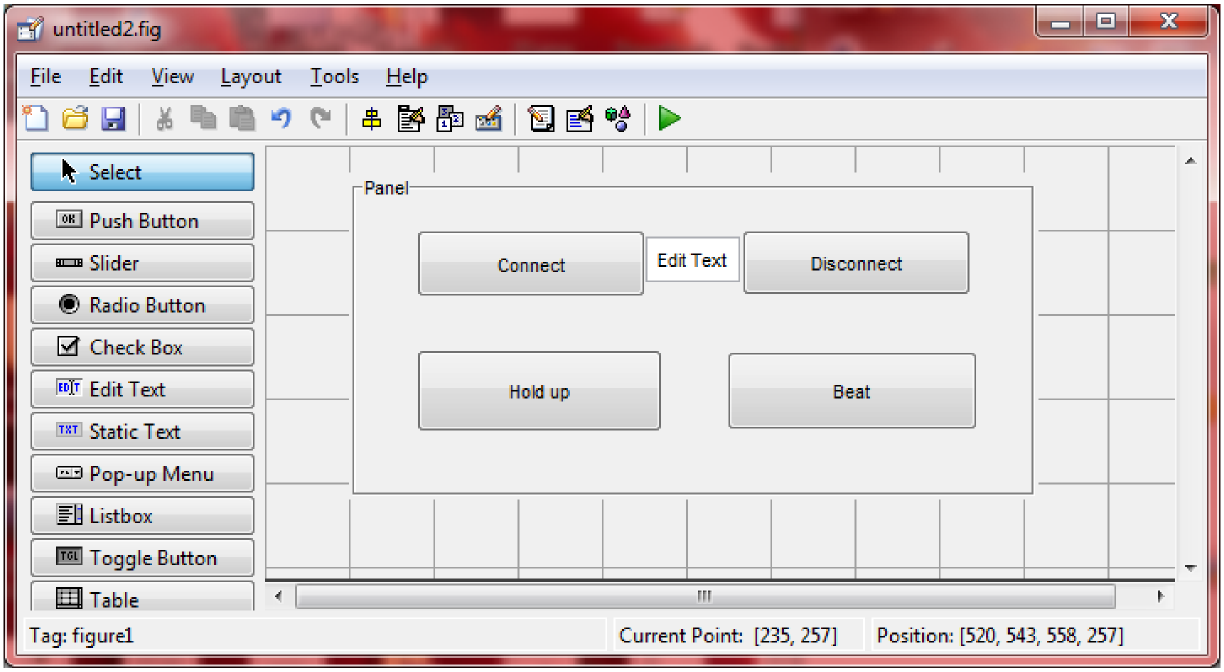


Figure.30 Mechanical Hammer System

Figure.30 shows how the mechanical system looks like.

5.8 Graphical User Interface Control



Tag: figure1 Current Point: [235, 257] Position: [520, 543, 558, 257]

Figure.31 Motor Driver Interface

Here we use MATLAB GUI to control this driver. The figure.31 shows the GUI interface. The detail of this software MATLAB will be described in Chapter 6.

Where the *Connect* button makes PC and driver connected, *Disconnection* button stops the controlling for the driver, *Hold up* button lets hammer go back for certain angle and *Beat* button lets hammer strike the Iron bar very quickly.

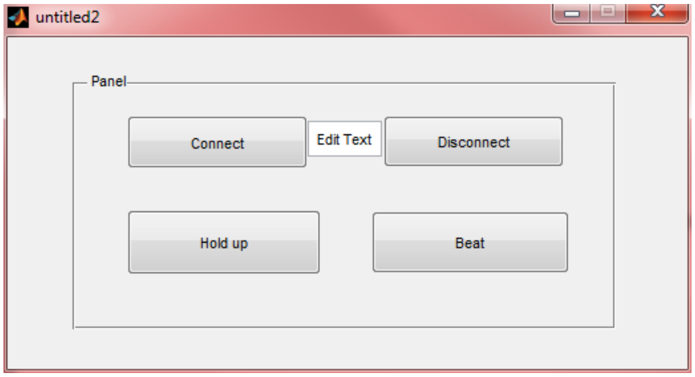


Figure.32 Final Control Panel

The ficture.32 shows how the final visual interface looks like and this is exactly what we will use to control the motor.

Chapter 6

Oscilloscope's PC controller

In this chapter one oscilloscope controlling method is presented. This is a classic method to send commands via RS232 serial port. The oscilloscope we used is Agilent 54622A, and then programming by using MATLAB. This method is very general, because it can control any oscilloscopes and equipment, even if they have no driver.

6.1 Introduction

This chapter presents how to build own oscilloscope driver. The Agilent 54622A has 100MHz bandwidth. It is connected with one PC and be controlled via the RS232 serial port. The oscilloscope is an analog/digital oscilloscope. It has an RS232 serial port and a microcontroller with certain commands given by product datasheet. Some command functions are not easy to implement without a computer, like saving a graph from the oscilloscope, or sending a graph to the oscilloscope. For the final visual panel programmed, it is better buttons have same name, same range and same function as those presented on the oscilloscope. Before controlling with PC, we must set the XON handshaking inside the I/O setting in order to match the PC setting.

6.2 MATLAB Overview

The proposal is to make a simply to control but efficient driver for the oscilloscope via RS232 interface.

The programming software used is MATLAB. This software implemented higher programming language is very simple and powerful. It also has graphical interface which will be used for our oscilloscope driver programming.

The disadvantage of MATLAB is that it only has Windows style controlling. By other words, if need changing voltage from MATLAB, have to apply slider instead of real rotary button.

The advantage of MATLAB is that any change on the graphical interface will immediately directly generate related MATLAB' m code file.

6.3 Graphical Interface Programming

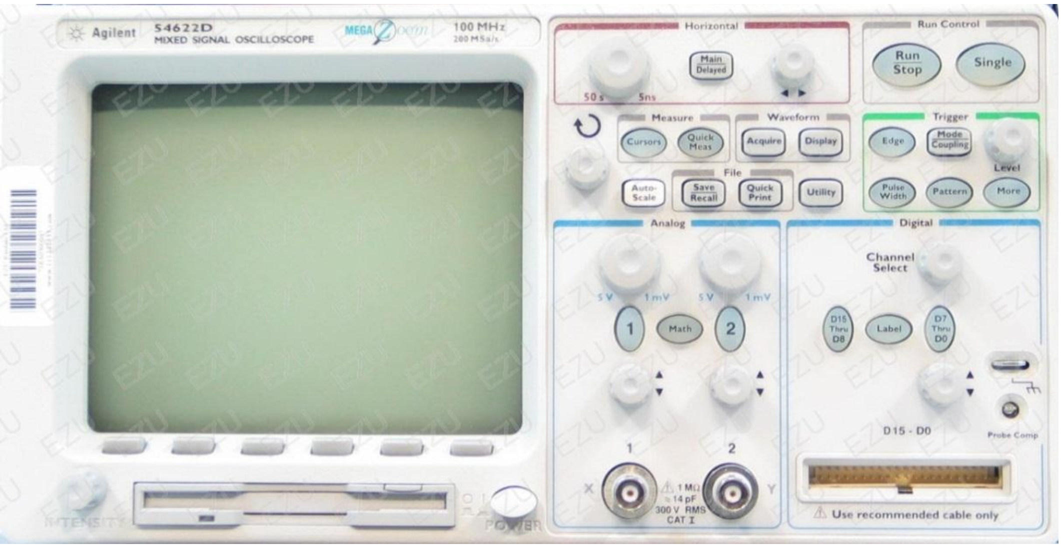
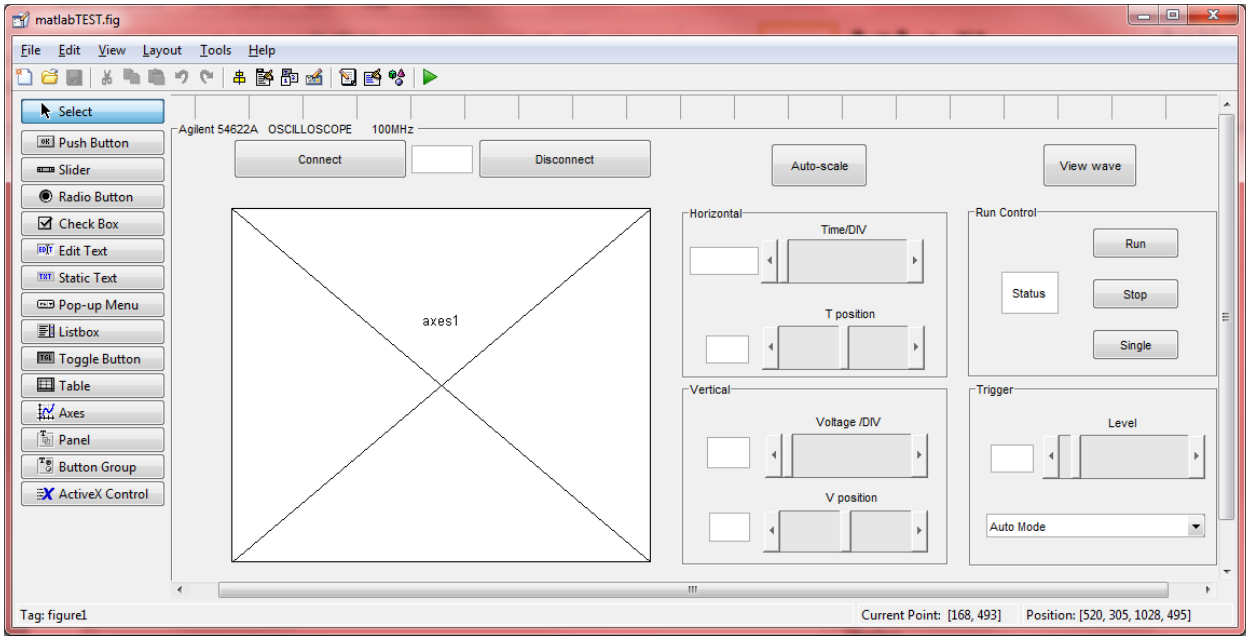


Figure.33 Agilent 54622 Oscilloscope

MATLAB graphical interface can be accessed when the *guide* command is typed at the command prompter. This command will open a window like the one in figure.34. Here the user can drag-and –drop buttons, sliders and other windows style controls and indicators needed.

In the oscilloscope interface sliders, Push buttons, pop-up menus controls and edit text, axes indicators were used. In the figure.34 we can see we have a *Connect* button for connecting the computer to the oscilloscope. This buttons sends the command, which puts the equipment in remote mode. During this time, on the graphical interface the text indicator at right of *Connect* button will shows *Connecting* after finish connecting. At the same time the *Remote* will also be shown on the screen of oscilloscope. Depending the state of the oscilloscope (remote or local), the text indicator also can be shown as *Disconnect* when they are disconnected by clicking *Disconnect b*utton.



Tag: figure1 Current Point: [168, 493] Position: [520, 305, 1028, 495]

Figure.34 MATLAB Graphical User Interface

The *Auto-scale* button from the user interface has the same effect as the Auto-scale button physically presented on the oscilloscope. It sets the Time/DIV and Volt/DIV parameters most optimal to the acquisitioned signal.

For setting the horizontal (Time-position) and vertical (Vol-position) of the signal, sliders were used.

For setting the timebase (Time/DIV) and the amplitude (Vol/DIV), also sliders were used. Since in the physical controlling oscilloscope, the range of variation is not linear changing, so we will use some ways to make the interface controlling as much same as physical controlling. And the detail of this part is described in the Appendix VI.

In the oscilloscope, there are two buttons named $\frac{Run}{Stop}$ and *Single* in the Run Control sub-window. So we need implement all functional buttons in the graphical interface. We created *Run*, *Stop and Single* buttons inside the Run Control sub-window of graphical interface.

For setting the trigger, we need set the level of trigger and select modes. There are two modes, *Auto mode* and *Normal mode*, and we create this selecting by adding a *Pop-up Menu* in the graphical interface.

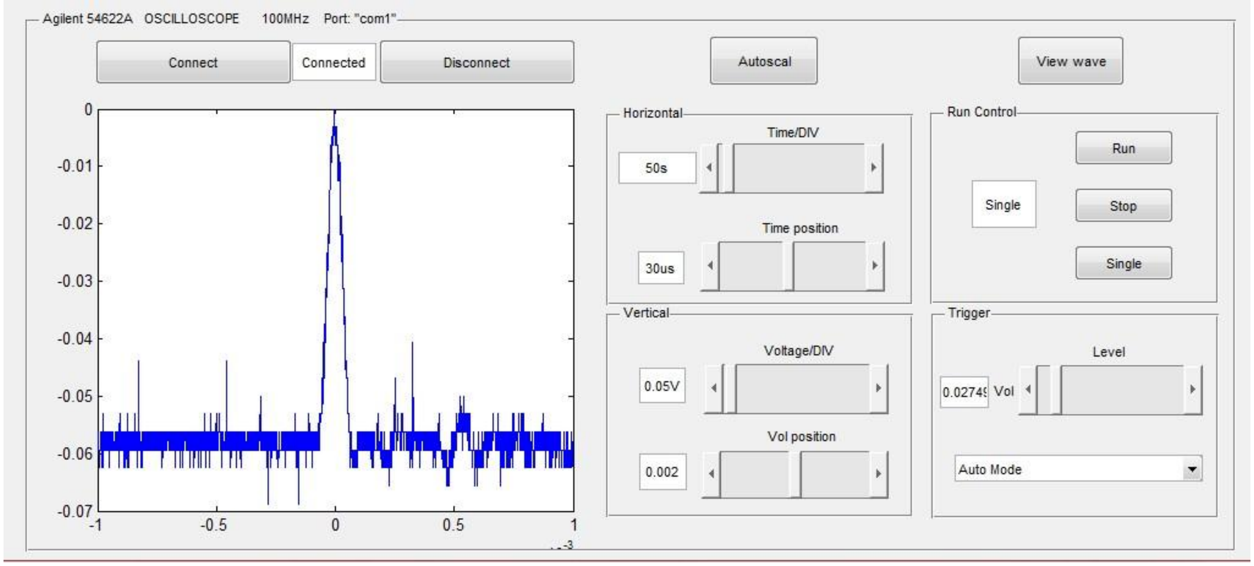
For writing the waveform from oscilloscope to PC, the *View wave* button was used. The waveform data we got from oscilloscope is captured by using a software which can return a good result with more sampling frequency. And this software is introduced in section 6.5.

The below table *Table 3* lists the commands used in programing this interface and these commands can be obtained from the oscilloscope user manual of this oscilloscope series.

Table 3 Used commands

Command	Description
:Autoscale	Auto setting for match input signal
:Timebase:range #	Sets Time division
:Timebase:position #	Sets Horizontal (Time position)
:Channel1:range #	Sets Voltage division
:Channel1:offset #	Sets Vertical (Voltage position)
:Run	Puts the oscilloscope into run mode
:Stop	Puts the oscilloscope into Stop mode
:Single	Puts the oscilloscope into Single mode
:Trigger:level #	Sets trigger level
:Trigger:sweep auto	Sets trigger mode into Auto
:Trigger:sweep normal	Sets trigger mode into Normal

The final front panel is shown as figure.35 which we will use to control oscilloscope totally by PC via RS232 cable. For waveform reading function, we need firstly capture the data of waveform and then plot into the axes of graphical interface as figure.35. The waveform data we captured is a CSV file which has two columns the one is x-axis another is y-axis. This two columns file format must be selected during capturing data via the software. The figure.36 shows the CSV file had 2000 sample points and it is selected during capturing data and plotted result is shown in figure.35.


Agilent 54622A OSCILLOSCOPE 100MHz Port: "com1"

Connected

Time/DIV: 50s

Time position: 30us

Voltage/DIV: 0.05V

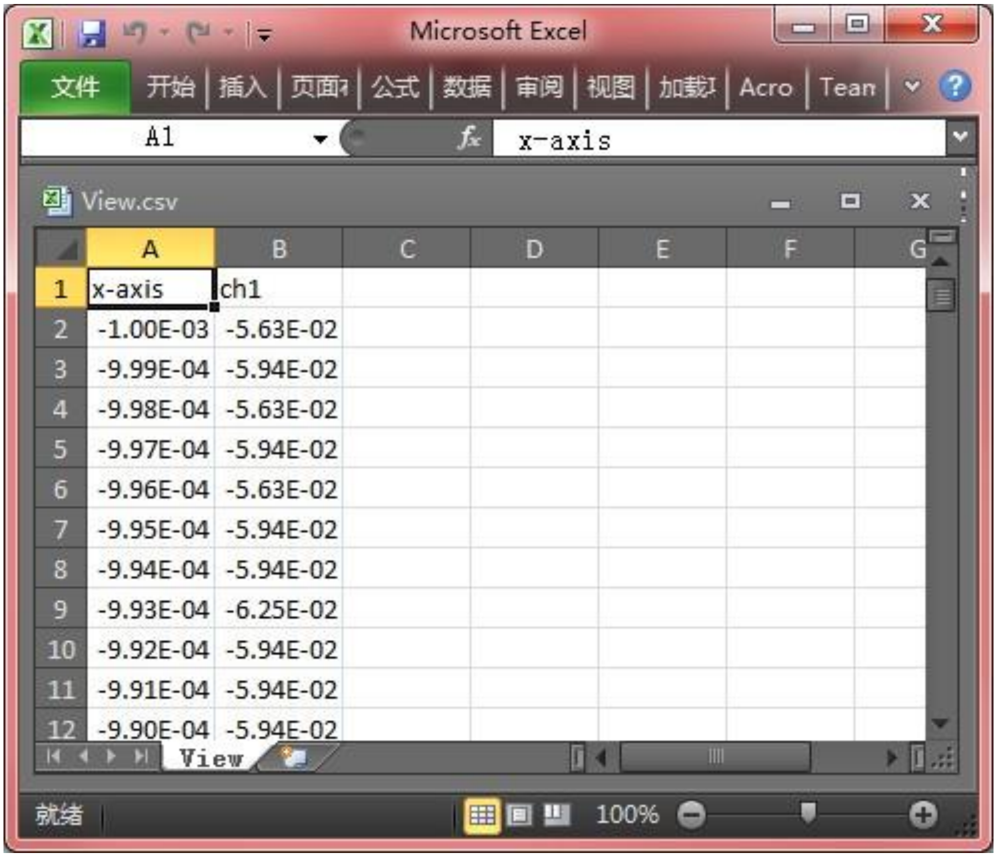
Vol position: 0.002

Single

Level: 0.0274! Vol

Auto Mode

Figure.35 Final front panel



A1 x-axis

	A	B
1	x-axis	ch1
2	-1.00E-03	-5.63E-02
3	-9.99E-04	-5.94E-02
4	-9.98E-04	-5.63E-02
5	-9.97E-04	-5.94E-02
6	-9.96E-04	-5.63E-02
7	-9.95E-04	-5.94E-02
8	-9.94E-04	-5.94E-02
9	-9.93E-04	-6.25E-02
10	-9.92E-04	-5.94E-02
11	-9.91E-04	-5.94E-02
12	-9.90E-04	-5.94E-02

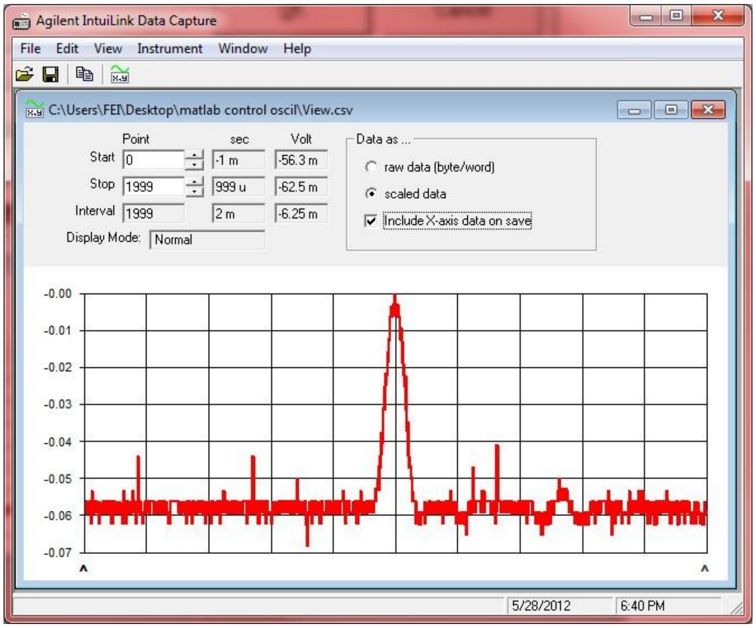
Figure.36 CSV waveform data

6.4 Waveform Data Capture

This section describes a software, IntuiLink Data Capture, which is designed by Agilent to capture data directly from equipment. It gives a convenient processing to do so.

The waveforms are actual time and voltage data from the oscilloscope. IntuiLink Data Capture displays them graphically, but they are stored as a table of time- voltage pairs. This data can be stored in several other formats of copied to other programs for further analysis such as MATLAB.

Remark: 1) Oscilloscope waveform data capture with IntuiLink Data Capture.


Agilent IntuiLink Data Capture
File Edit View Instrument Window Help
-0.00
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
-0.07

C:\Users\FEI\Desktop\matlab control oscil\View.csv

	Point	sec	Volt
Start	0	-1 m	-56.3 m
Stop	1999	999 u	-62.5 m
Interval	1999	2 m	-6.25 m

Display Mode: Normal

Data as ...
○ raw data (byte/word)
● scaled data
☑ Include X-axis data on save

5/28/2012 6:40 PM

Figure.37 LintuiLink Data Capture

Remark: 2) Oscilloscope waveform got via camera.

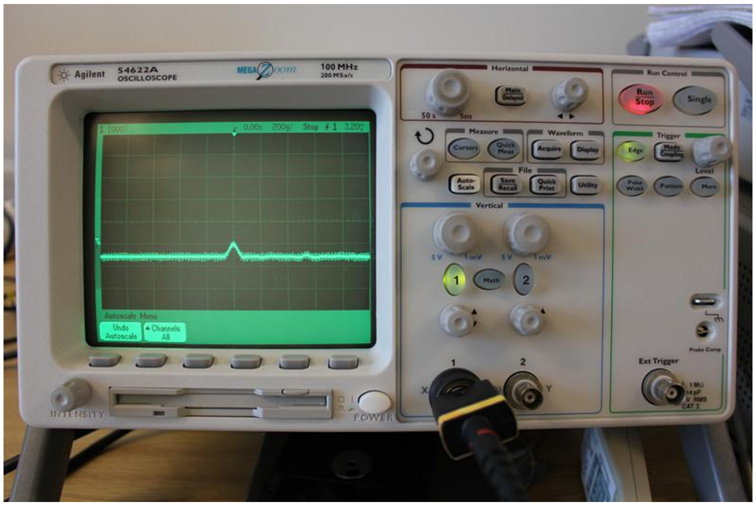


Figure.38 Waveform on the Oscilloscope

The Agilent Data Capture is a dependent program for the purpose of downloading waveform data from Agilent Oscilloscopes. It provides the following functionality:

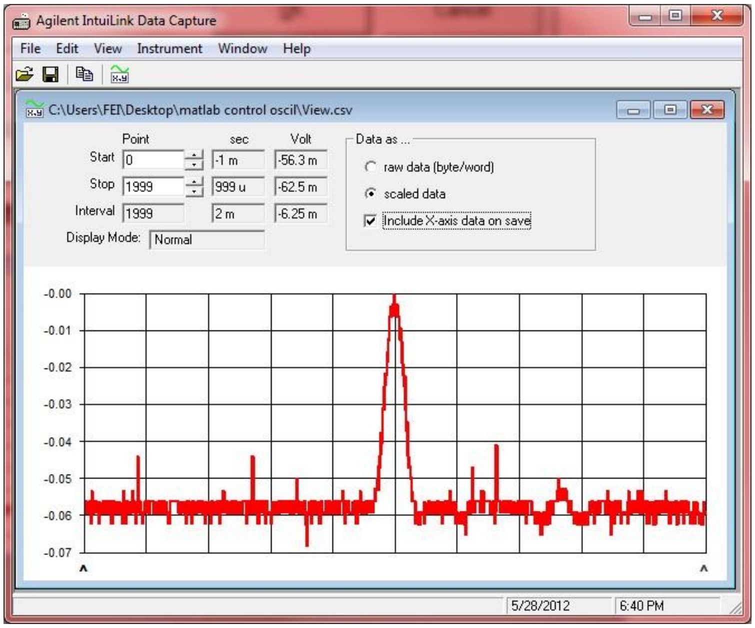
- Download waveform data and display the data as a simple image
- Save the data as binary or text files
- Load saved data back into the application

Connecting to the Instrument

It needs few steps to connect this software to the oscilloscope in order to read wave. And this procedure is introduced in the Appendix I in detail. So you can only follow those steps to finish connection.

Display of Waveform Data:

After complete the connection between the software and oscilloscope, the wave will directly be shows in window just as same as wave in the oscilloscope as figure.39 shown.



C:\Users\FEI\Desktop\matlab control oscil\View.csv

	Point	sec	Volt
Start	0	-1 m	-56.3 m
Stop	1999	999 u	-62.5 m
Interval	1999	2 m	-6.25 m

Display Mode: Normal

Data as ...
○ raw data (byte/word)
● scaled data
☑ Include X-axis data on save

5/28/2012 6:40 PM

Figure.39 Waveform with setting scaled data

Save to a File

Since the wave reading function we programmed in the user’s interface just plots a CSV file. So we have to save data to a file that can be read by MATLAB. The format has lot of types such as Text (Tab delimited), Text (space delimited), CSV, and Binary. But we need to save as CSV file in this project in order to be plot by the MATLAB interface.

Chapter 7

Measurement of Pressure wave

In order to read the strain gauge changing, we need to use related amplifier to amplify this signal. Since we have introduced the strain gauge sensor in section 1.3 and section 3.3.1, in this chapter the amplifier connection will be described. The figure.41 shows the strain gauge sensor glued on the iron bar. We can see that there are four wires out coming from the sensor that will be connected with sensor amplifier.

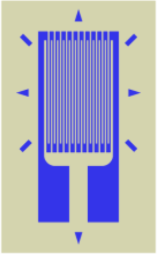
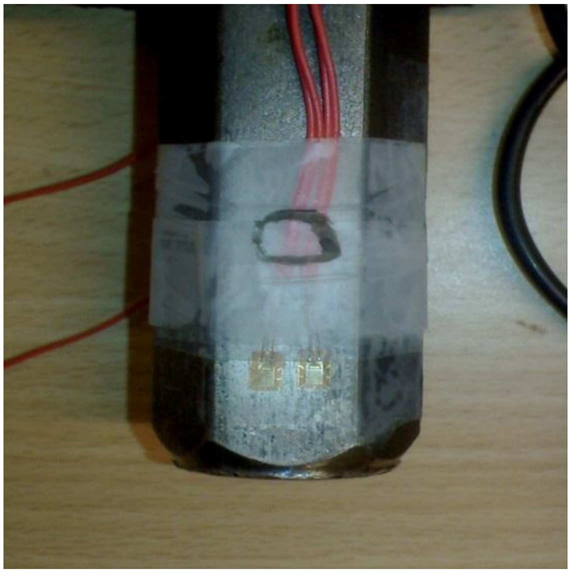


Figure.40 Strain gauge sensor



.

Figure.41 Strain gauge sensor installed on the Iron bar

7.1 Strain Gauge Amplifier

The board used in the project is named Strain Gauge Pre Amp Board with 9- pin D- Type Cable. It used to connect with the strain gauge sensor. The figure.42 shows this amplifier.

This pre-amplification circuitry is designed to amplify resistance changes within a full Wheatstone bridge strain gauge, to produce a signal suitable for measuring in oscilloscope.

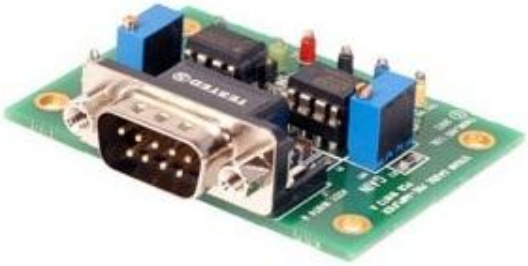


Figure.42 Strain gauge amplifier

The amplification circuitry uses a low power, general purpose instrumentation amplifier offering excellent accuracy suitable for strain gauge bridge amplification. Current-feedback input circuitry provides wide bandwidth even at 200KHz.

7.2 D-type connections

The pin out of the D-type connector is designed for directly connecting to the THORLABS's product, but in this project need to connect it with a normal oscilloscope so we cut the wire then matching each wire with each color just as figure.44 illustrated.

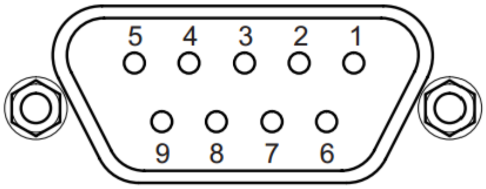


Figure.43 Ports of the Cable

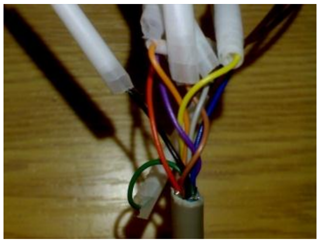


Figure.44 Ports inside the Cable

Table 4 Pin description

Pin Number	Input/ Output
Pin 1 (Brown)	1Voltage Oscillator input
Pin 2 (Red)	+15 input Supply
Pin 3 (Orange)	- 15 input Supply
Pin 4 (Yellow)	0V Supply (Ground)
Pin 5 (Green)	Output signal
Pin 6 (Blue)	0V Supply (Ground)
Pin 7 (Violent)	ID Resistor Connection (Ground)
Pin 8 (Grey)	N/C
Pin 9 (Black)	N/C

By following Table 4, we connect these wires with power supply and have a correct connection way that can use oscilloscope to measure the output signal from this filter. Connection details are shown as below figure.45.

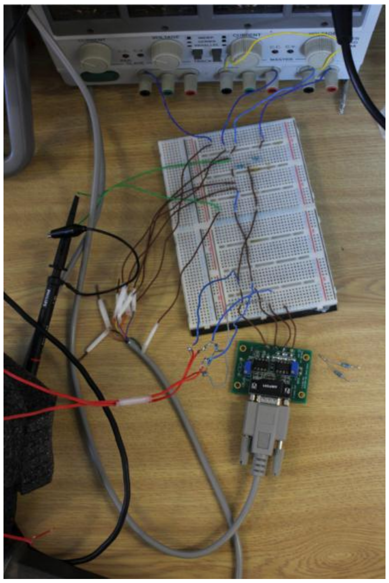


Figure.45 Amplifier connection with related input or output

7.3 Strain Gauge Bridge Connections

Connections of the full Wheatstone bridge amplifier circuit are shown in detail in figure.46.

Table 5 Wheatstone bridge connection

Color Port	Input / Output
Blue port:	Positive output of strain gauge bridge (Same port with Pin 1)
Yellow port:	Negative output of strain gauge bridge
Red port:	Oscillator input of strain gauge bridge
Black port:	Ground

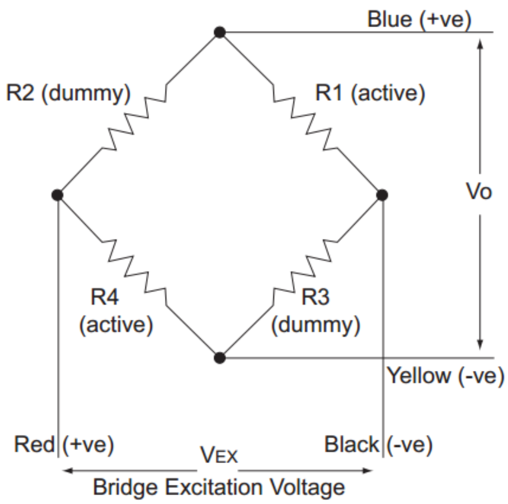


Figure.46 Full Wheatstone Bridge Connection

According to this Wheatstone bridge connection as figure.46 and the Table 5, we can see that R1(active), R4(active) both are Strain Gauge sensors attached on the Iron bar. So we give the R2, R3 as normal resistor 350Ω respectively. (The sensor resistance is 350 Ω and the dummy's resistance must be as same as active's resistance)

The Red port is an oscillator signal receiving port. We give signal as 1 voltage supply. Since considering certain equipment, there is only one supply which can generate 5V fixed voltage so the voltage divider has to be introduced. The circuit is designed as figure.47.

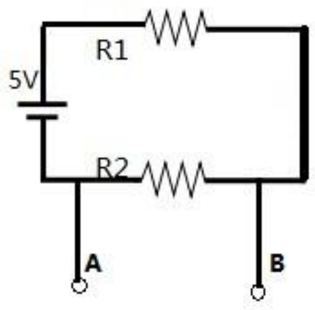


Figure.47 Voltage divider circuit

Since we want to have a 1V output from this voltage divider, the rate of R_1 and R_2 should 4:1(R_1 =4R_2). The two terminals A,B will be as a oscillator supply for the wheatstone bridge, so the load must be considered. But the value of load's resistance is unknown. We have to try it by several times to decide which R_2 is suitable. The basic method is that R_2 should be small enough compared with load. Finally R_2 is decided as 60Ω and R_1 is 240 Ω. So the Red port and Black port will be connected with two terminals A and B.

Chapter 8

Conclusions

This project is mainly created for distance lab. Using those programmed software you can perform every things just like doing this experiment physically. The experiment can be performed at any time any place and students will don't have to go to Lab room. This way will protect hardware not be destroyed and also save time for students and teachers. This distance lab system is mainly analyzing the pressure wave propagation in an iron bar.

This report introduces some equipment selection especially strain gauge sensor and the connection between different equipment and also introduces how to install this system and how to use this system. In terms of software, this report describes some related programming which is used to let hardware (oscilloscope and motor) be controlled by a PC via internet. Finally the completed interface is created for controlling whole system and this visual controlling panel has high speed of communication between a PC and equipment. The modeling represented in the beginning is used to analysis the real physical system by a theoretical way. This is also a typical way that used to check the physical experiment to see if there is any error existed.

Now we do not only have the complete distance lab, but also have affiliated software to control it and theoretical method supporting the related calculation and related working procedure. So this report can be used by who is very interesting in this distance system.

References

[1] O.Thomas Holland and Patrick Marchand, Graphics and GUIs with Matlab, Third Edition (Graphics&GUIs with MATLAB). 25 Nov 2002

[2] A.L. Window, Strain Gauge Technology, Springer, 2nd ed. Edition(30 Nov 1992), ISBN-13: 987-1851668649

[3] “Principles of Measurement Systems” John P. Bentley. Publisher: Prentic Hall; 4 edition. ISBN-10: 0130430285

[4] “Modeling of Dynamic Systems” Lennart Ljung and Torkel Glad. Publisher: Prentice Hall; 1 Edition (May 5, 1994). ISBN-10: 0135970970

[5] Arik D. Brown. Electronically Scanned Arrays MATLAB Modeling and Simulation. CRC Press; 1 edition. 8 May 2012 ISBN-10: 1439861633

[6] Howard W. Johnson and Martin Graham. High-Speed Signal Propagation. Prentice Hall; 1 edition 24 Feb 2003.

[7] Giovanni Bianchi. Electronic Filter Simulation& Design. McGraw-Hill Professional; Har/Cdr edition 1 July 2007. ISBN-10: 0071494677

[8] Harprit Singh Sandhu. Running Small Motors with PIC Microcontrollers. McGraw-Hill professional;1 edition 1Aug 2009. ISNB-10:0071633512.

[9] DAVID G. ALEXANDER and R.E.SMELSER, Delivering an Engineering Laboratory Course Using the Internet, the Post Office, and a Campus Visit.

[10] www.bigtopmania.co.uk/pdf_downloads/HSE_Guidance/hss_guide_to_wacker_use.pdf

[11] http://hyperphysics.phy-astr.gsu.edu/hbase/sound/souspe2.html

[12] Foss,B,A,Malvig,K.E.,Eikaas, T.T., ”Remote Experimentation- New Content in Distance Learning”, Proceedings of the ICEE 2001 Conference in Oslo, Norway, August 6-10,2001.

[13] Benson, H., University Physics, John Wiley & Sons Inc, 1991, pp41-42.

[14] Johnson D., Johnson J., Hilburn J., Scott P., Electric Circuit Analysis, Third edition, Prentice Hall International, Inc., 1997.

[15] “Remote laboratory experiments in electrical engineering education” Gustavsson, I. *(*Dept. of Telecommun. &Signal Process., Blekinge Inst. Of Technol., Ronneby, Sweden) *Source: Proceedings of the Fourth IEEE International Caracas Conference on Devices, Circuits and system(Cat. No.02TH8611), p I025-1-5, 2002.*

[16] “A remote access laboratory for electrical circuit experiments” Gustavsson, I. (Dept. of Telecommun. & Signal Process., Blekinge Inst. Of Technol., Sweden) Source: *International Journal of Engineering Education*, v19, n3, p409-19,2003.

[17] “Remote operation and control of traditional laboratory equipment” Gustavsson, I. (Dept. of Singal Process., Blekinge Inst. Of Technol., Ronneby, Sweden); Zackrosson, J.: Akesson, H.; Hakansson, L.; Claesson, I.; Lago, T. Source: *International Journal of Online Engineering*, v2, n1, p8 pp., 2009.

[18] "Anytime, Everywhere- Approaches to Distance Labs in Embedded Systems Education" Markus Proske and Christian Trödhandl, (Vienna University of Technology, Institute of Computer Engineering, Vienna, Austria)

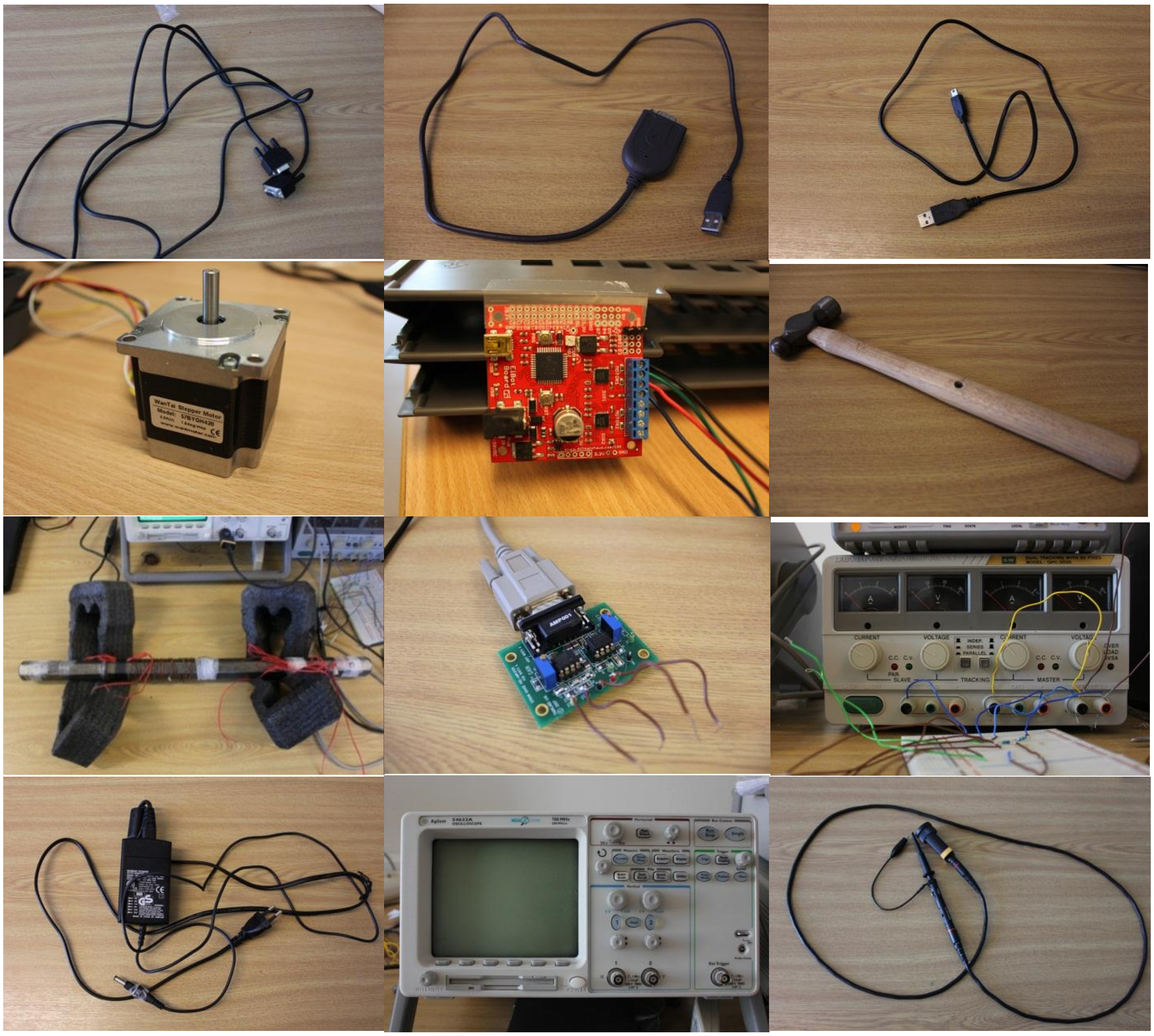
[19] "Development of Distance Real laboratory System" Kazutake Kozono, Hidenori Akiyamma and Naoyuki Shimomura. (Graduate School of Science and Technology, Kumamoto University, Department of Electrical and Electronic Engineering, Tokushima University.)

[20] Atlas Copco company's breaker http://www.atlascopco.se/sesv/

Appendix I

Distance Lab’s User Manual

This lab consist of one RS232 cable, one RS232-USB converter, one USB cable, one stepper motor, one EiBotBoard, one hammer, one iron bar glued with strain gauge sensor, one Strain Gauge Pre Amplifier, one power supply, one 15voltage power supply, one Agilent 54622A Oscilloscope, one oscilloscope probe, one PC. Below pictures list these elements.

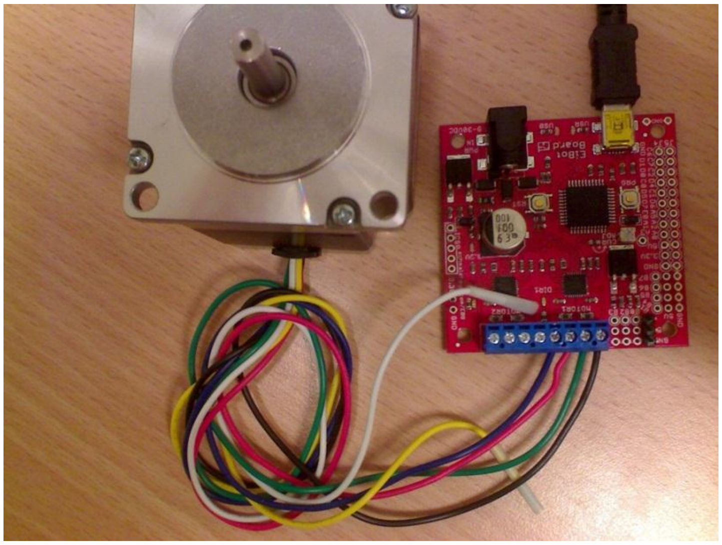


Appendix fig.1 Distance lab equipment

Physical system connection

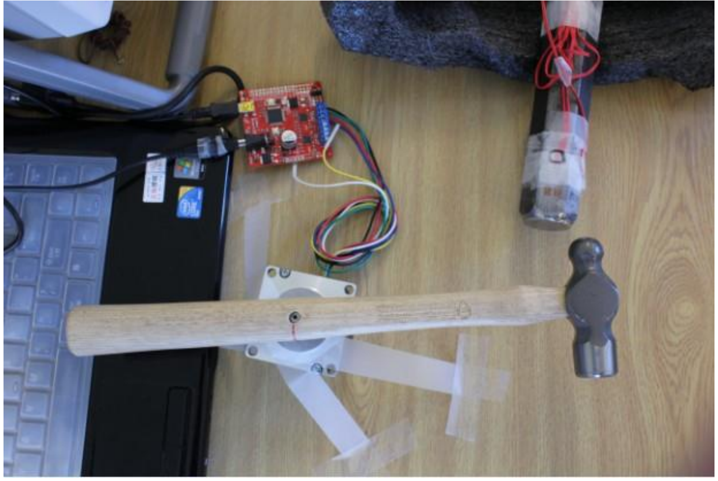
First we connect these elements as one entire system.

- Connect oscilloscope RS232 port with RS232 cable then connected with USB converter finally insert into PC’s USB port.
- Connect EiBotBoard with 15voltage power supply and USB cable then insert USB cable into PC’s USB port. Also connect EiBotBoard with stepper motor as below picture shown.



Appendix fig.2 Motor connection

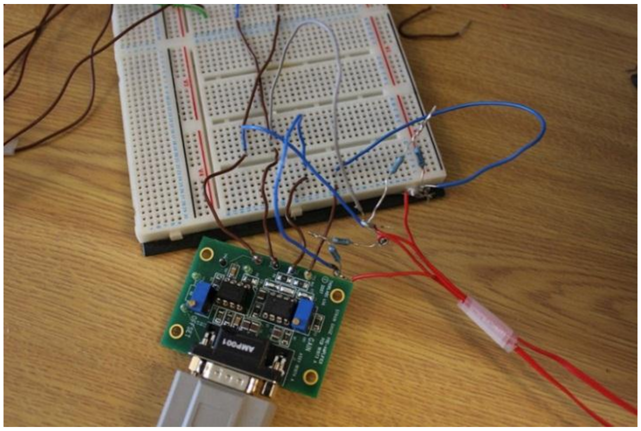
- Insert the hammer onto stepper motor as below picture shown.



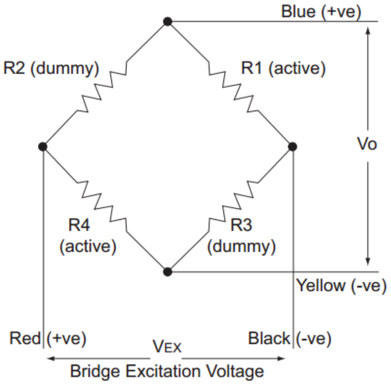
Appendix fig.3 Hammer connected with motor

- Coordinate the hammer with iron bar that the hammer can hit iron bar’s end glued strain gauge sensor perfectly.

- Connect Strain Gauge Pre Amplifier with strain gauge as below appendix fig.3 shown. Where the four red wires are coming from strain gauge. The connection follows the below appendix fig.4.



Appendix fig.3 Strain gauge amplifier connection



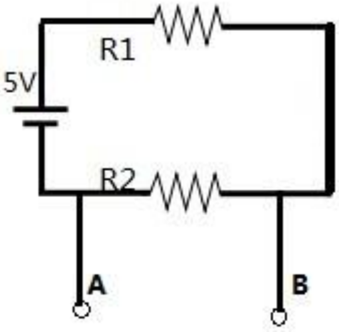
Appendix fig.4 Connection description

Where in the right picture, R1 and R4 are strain gauge sensor and R2,R3 are two 350 Ω resistor respectively.

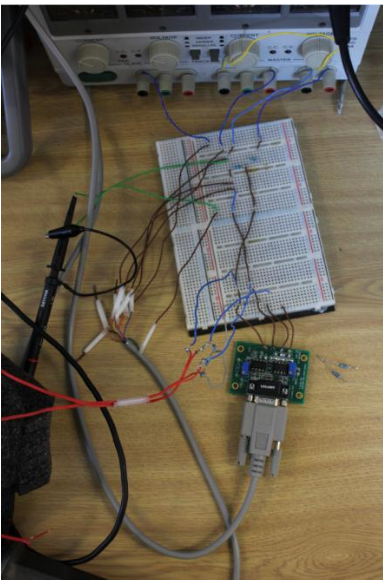
- Give relative input of Strain Gauge Amplifier by below list.

Pin 1(Brown):	Oscillator input	(Blue port)
Pin 2(Red):	+15 input Supply	(+15V input)
Pin 3(Orange):	- 15 input Supply	(-15V input)
Pin 4(Yellow):	0V Supply	(Ground)
Pin 5(Green):	Amplifier Output	(Output signal)
Pin 6(Blue):	0V Supply	(Ground)
Pin 7(Violent):	ID Resistor Connection	(Ground)
Pin 8(Grey):	N/C	
Pin 9(Black):	N/C	

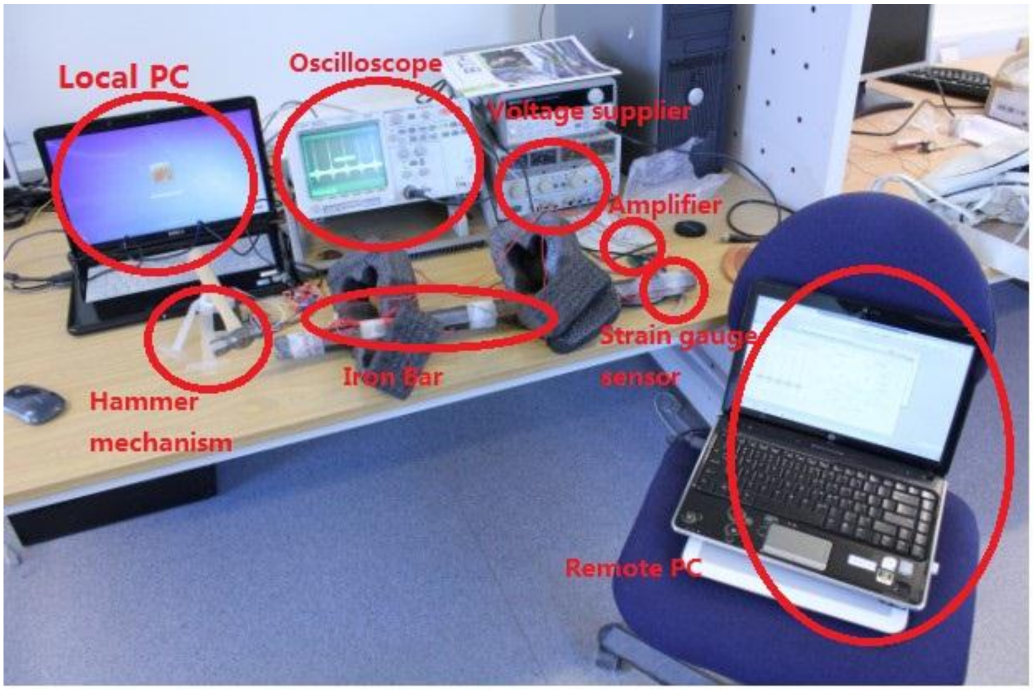
- Give 1 voltage input (port A and B) to Pin 1(Oscillator input) and ground by the voltage divider as following picture shown. Where R1 is 240 Ω and R2 is 60 Ω.



- Connect oscilloscope probe with Pin 5. The final connection of Strain gauge Pre Amplifier is shown as below picture.

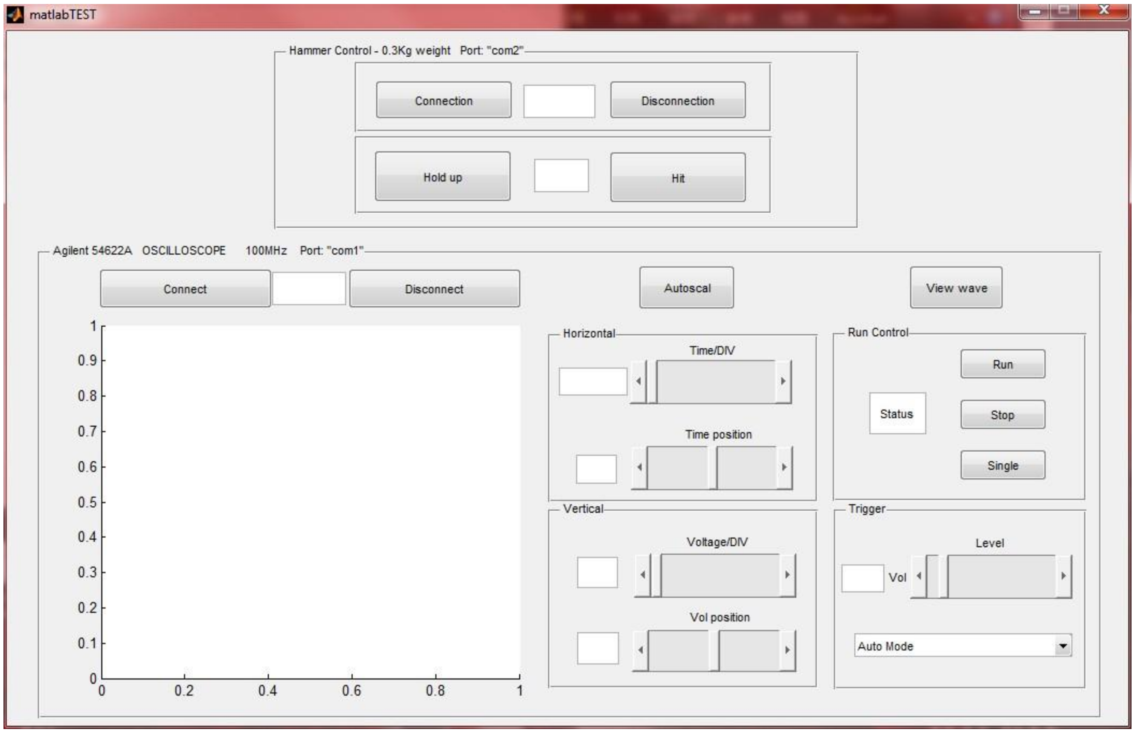


Until here the physical connection is complete and below picture shows the completely system connection.

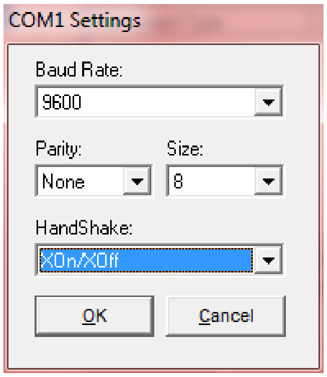


Software operating

Now turn on all powers and then open the interface of this system as below：



- Click *Connect* to active oscilloscope operation and then set the relative parameter as you want.
- Click *Connection* to active mechanism system.
- Click *Hold up* to make the hammer have a center angle with iron bar.
- Click *Hit* to let the hammer hit the iron bar to finish pressure wave generating.
- Before open IntuiLink Data Capture, click *Disconnect* to disconnect oscilloscope with PC
- Open IntuiLink Data Capture. From the *Instrument* menu, choose 54620 serial.
- click *set I/O* then click on *Find Instrument*… to initiate a search.
- Select which address the instrument currently used.
- Click *Identify Instrument(s)*. All the parameters should be as same as set in the oscilloscope, especially *HandShake* should set as *XOn/XOff*. Then press OK.
- Now your oscilloscope should be shown at right blank. After click *Connect*, a green icon appears to the left of the instruments that is connected. Then press OK

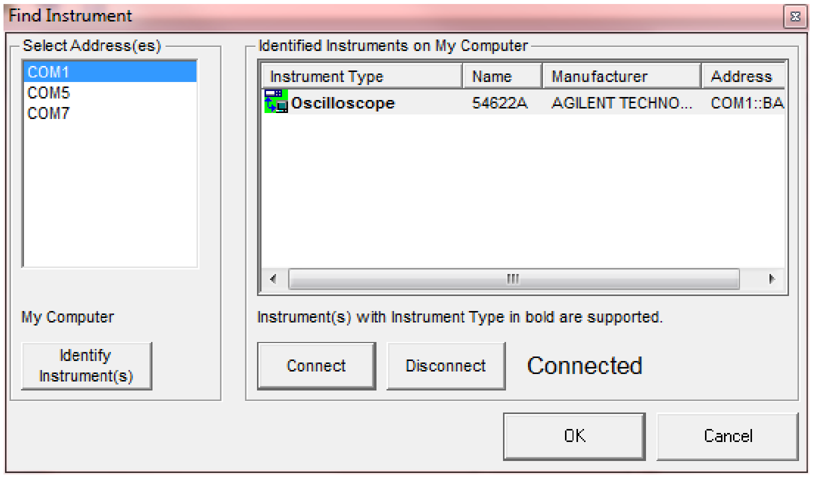

COM1 Settings

Baud Rate: 9600

Parity: None

Size: 8

HandShake: XOn/XOff



Find Instrument

Select Address(es): COM1, COM5, COM7

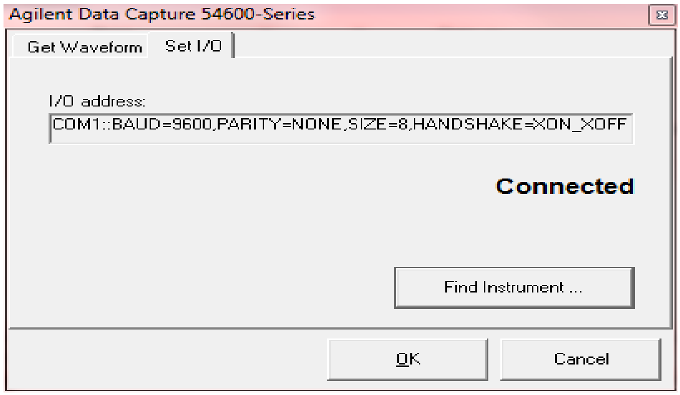
Identified Instruments on My Computer

Instrument Type	Name	Manufacturer	Address
Oscilloscope	54622A	AGILENT TECHNO...	COM1::BA

Instrument(s) with Instrument Type in bold are supported.

Connected

- The I/O address is changed for current data, just click OK to finish.

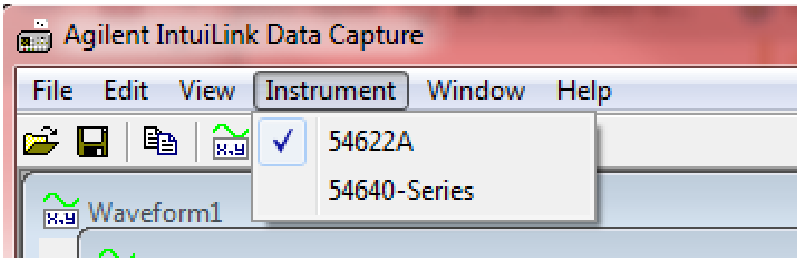


Agilent Data Capture 54600-Series

I/O address: COM1::BAUD=9600,PARITY=NONE,SIZE=8,HANDSHAKE=XON_XOFF

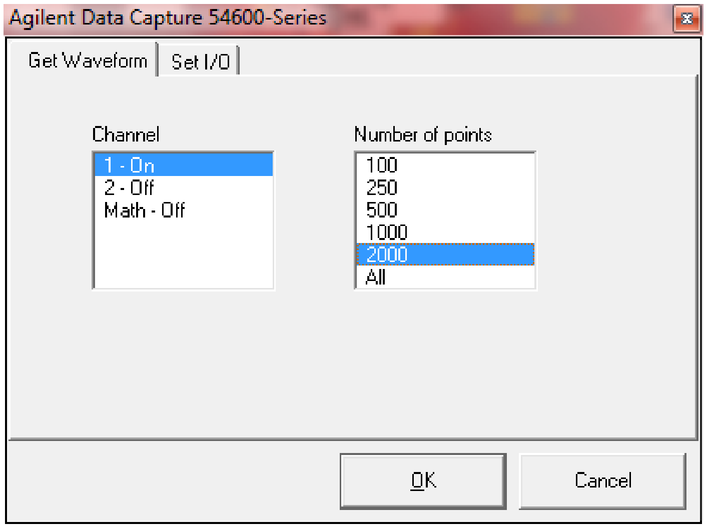
Connected

- To download data form the instrument, select from the instrument *menu* and select your instrument (or select the *'Get Waveform'* icon after Connected to the Instrument)



Agilent IntuiLink Data Capture

- Select the *channel* and *Number of points* wanted to download.



Agilent Data Capture 54600-Series

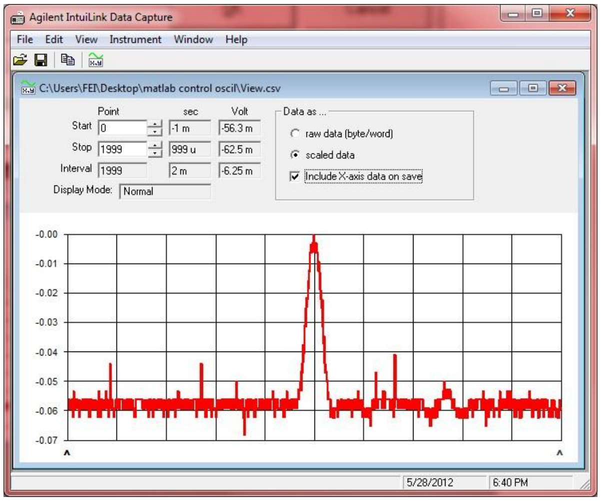
Channel

1 - On
2 - Off
Math - Off

Number of points

100
250
500
1000
2000
All

- Click *Include X-axis data on save* then save data to a file that can be read by MATLAB, CSV file that named “View.csv”. The saving path must belong to the file where the system’s interface in.



Agilent IntuiLink Data Capture

C:\Users\FEI\Desktop\matlab control oscil\View.csv

	Point	sec	Volt
Start	0	-1 m	-56.3 m
Stop	1999	999 u	-62.5 m
Interval	1999	2 m	-6.25 m

Display Mode: Normal

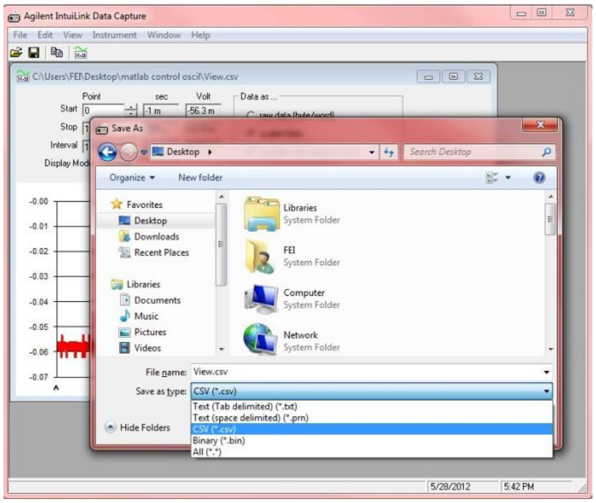
Data as ...

○ raw data (byte/word)

● scaled data

☑ Include X-axis data on save

5/28/2012 6:40 PM



Agilent IntuiLink Data Capture

C:\Users\FEI\Desktop\matlab control oscil\View.csv

Save As

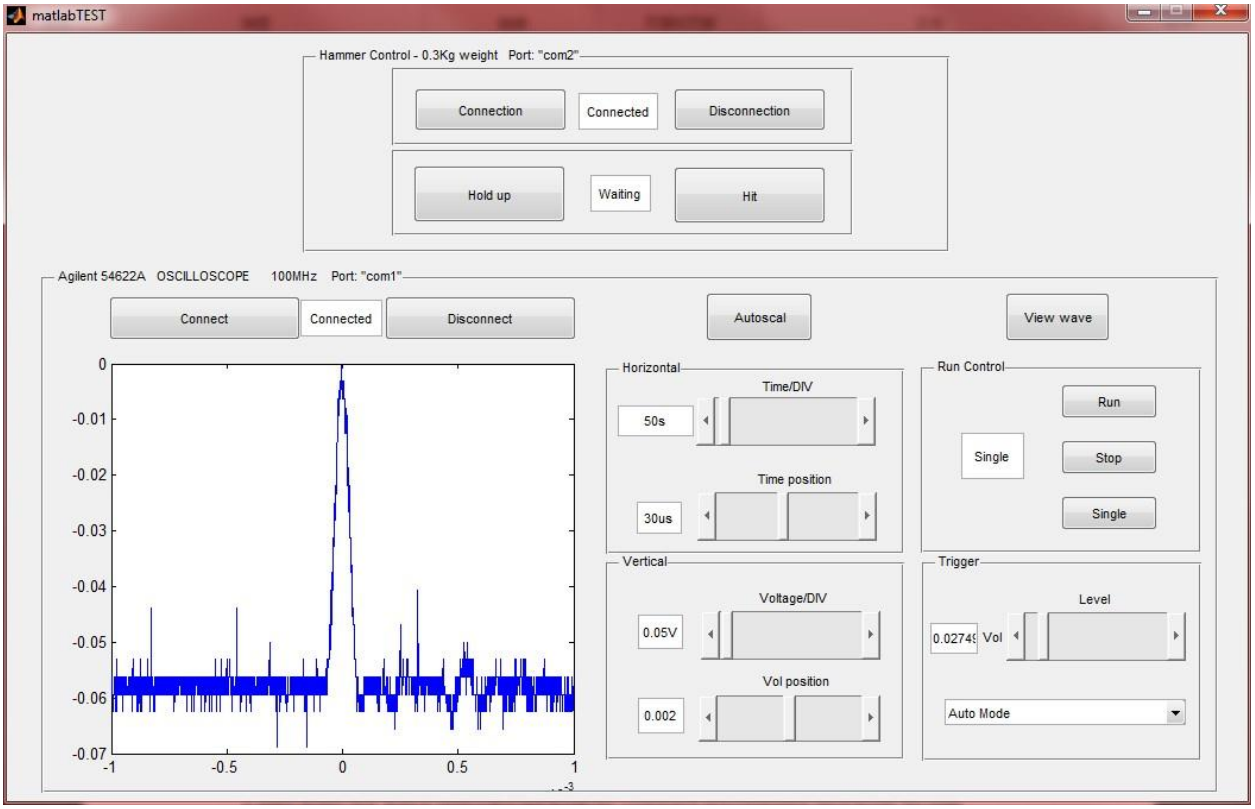
File name: View.csv

Save as type: CSV (*.csv)

Text (Tab delimited) (*.txt)
Text (space delimited) (*.prn)
CSV (*.csv)
Binary (*.bin)
All (*.*)

5/28/2012 5:42 PM

- To show this wave in MATLAB interface, just click *View wave* button in the interface. It actually just plot the CSV file “View.csv”



Single

Time/DIV 50s

Time position 30us

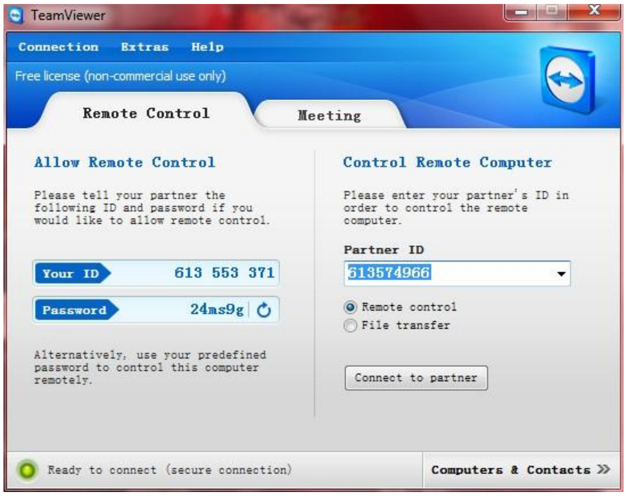
Voltage/DIV 0.05V

Vol position 0.002

Level 0.0274! Vol

Auto Mode

Until here we have introduced how to connect equipment and how to use interface to control this system. All the controlling can be operated by distance way. You only need have a PC to access the local PC by internet. There are a lot of software can do this and what we used is named ‘TeamViewer ‘. This can be free downloaded from internet. Both local PC and remote PC have to install this software and it will give your ID and password that you can use it to access the local PC. Then simply click *Connect to partner* to finish remote controlling.



Free license (non-commercial use only)

Allow Remote Control

Please tell your partner the following ID and password if you would like to allow remote control.

Your ID 613 553 371

Password 24ms9g

Alternatively, use your predefined password to control this computer remotely.

Control Remote Computer

Please enter your partner's ID in order to control the remote computer.

Partner ID 613574966

● Remote control

○ File transfer

Connect to partner

Ready to connect (secure connection)

Computers & Contacts »

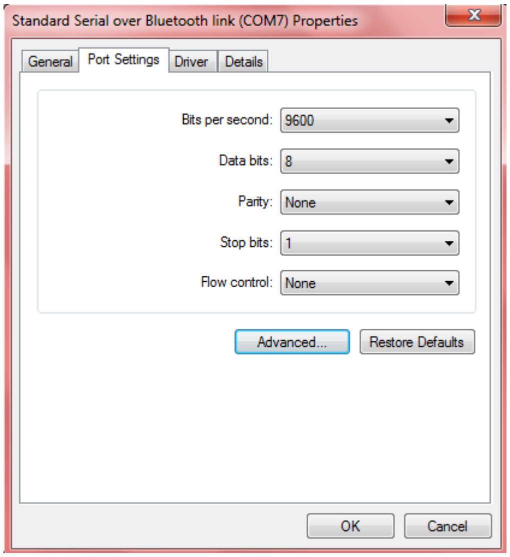
Following above description you can install all the equipment and then can access this distance lab in anywhere.

Appendix II

Software Setting of EiBotBoard

After install the UBW driver, we need to see or change the port number by following below steps.

Go to " *Start* " >> " *Control panel* " >> " *Hardware and Sound* " >> " *Device Manager* " >> Extend " *Port (COM&LPT)* " (You will see your UWB device inside this tree as “ “) >> Right click UWB device and choose " *Properties* " >> select " *Port Settings* " and " *Advanced..* " >> Now you can change the COM port number directly. Below figures show some steps of this port number changing.


Standard Serial over Bluetooth link (COM7) Properties

Bits per second: 9600

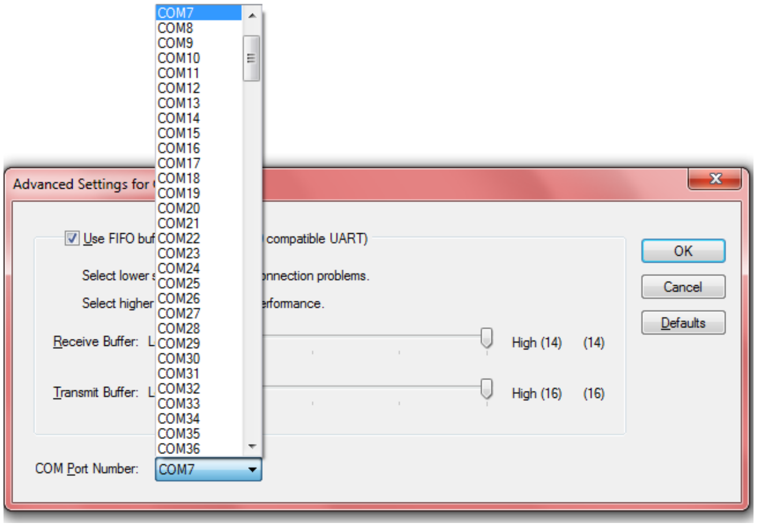
Data bits: 8

Parity: None

Stop bits: 1

Flow control: None

Figure AppII.1 COM Port Properties


Advanced Settings for

☑ Use FIFO buf compatible UART)

Select lower s onnection problems.

Select higher erformance.

Receive Buffer: L High (14) (14)

Transmit Buffer: L High (16) (16)

COM Port Number: COM7

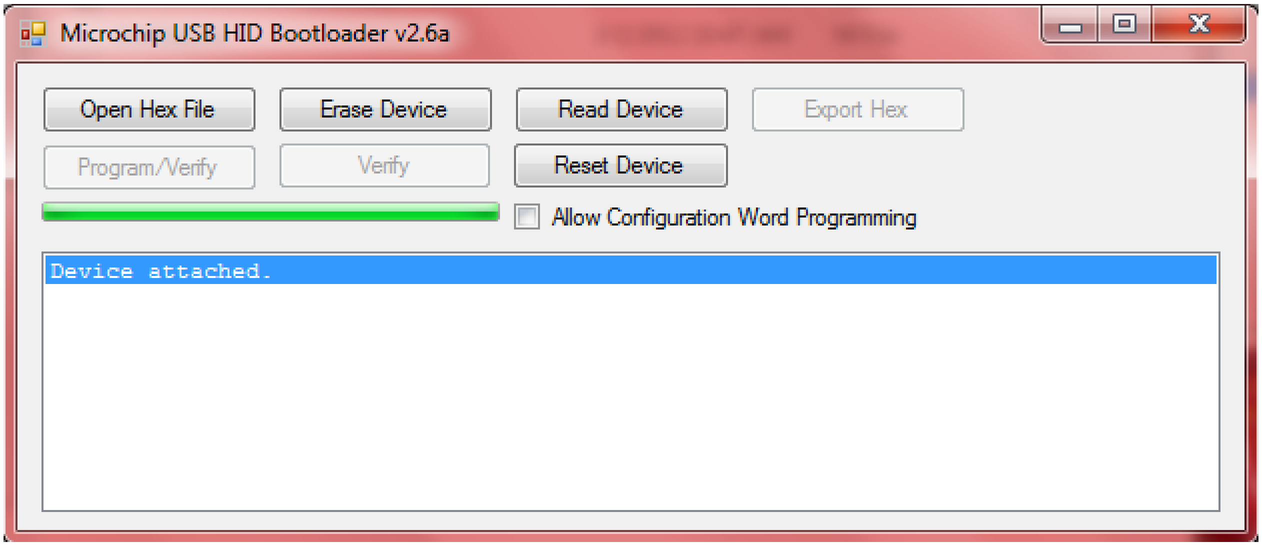
Figure AppII.2 Port number selection

Appendix III

Updating EBB firmware

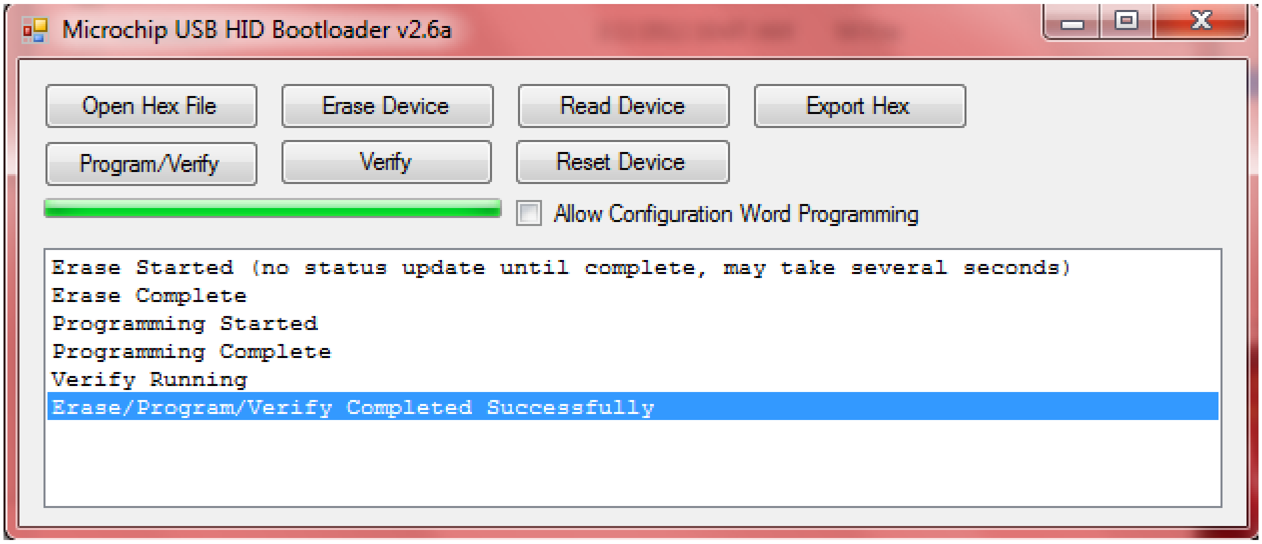
- Power on the EBB with the power supply, and connect the USB cable to your PC
- Press and hold the PRG button while pressing and releasing the RST button, then release the PRG button
- Run the HIDBootloader.exe program
- You should see it says "Device attached"
- Click Open Hex File, and select the HEX file that you want to download
- Click Program/Verify
- When programming is complete, click Reset Device
- When the Inkscape extension finds your EBB, it will tell you the firmware version number, so you can verify that the new version is detected properly.

Some interim results are shown below:



```
Device attached.
```

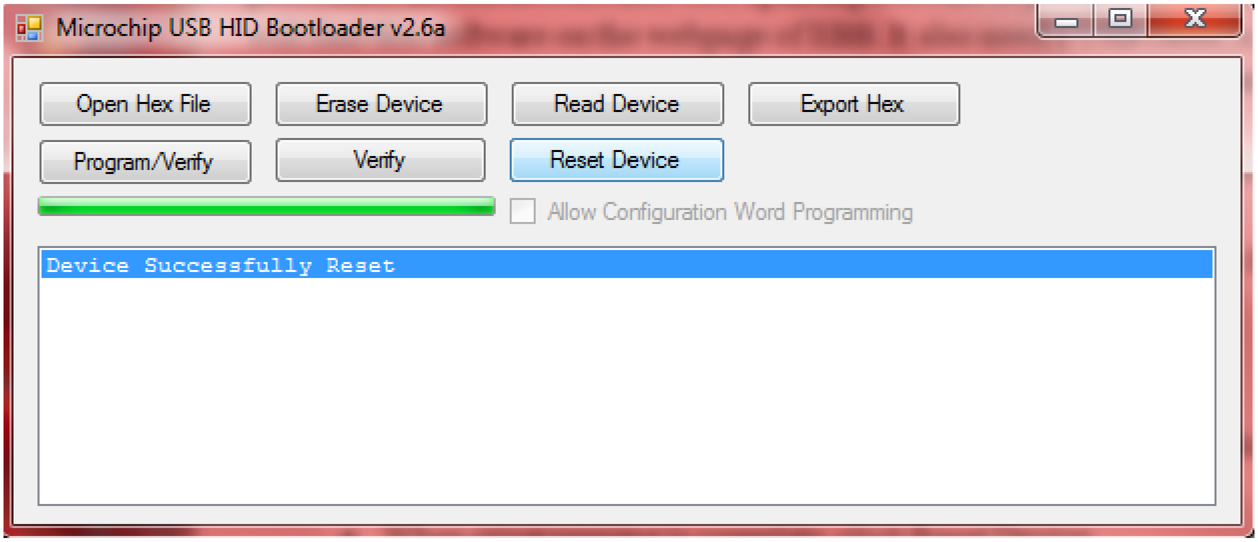
Figure AppIII.1



□ Allow Configuration Word Programming

```
Erase Started (no status update until complete, may take several seconds)
Erase Complete
Programming Started
Programming Complete
Verify Running
Erase/Program/Verify Completed Successfully
```

Figure AppIII.2



□ Allow Configuration Word Programming

```
Device Successfully Reset
```

Figure AppIII.3

Appendix IV

Test EBB board with Tera Term Pro

In order to test Section 5.5's command and if the board received the commands successfully, we use *Tera Term Pro* , a software that can send command to equipment via serial port, to send those commands. The procedure is going to be represented step by step.

- Connect USB cable and you can see the USB LED is light. Fig AppIII.1.

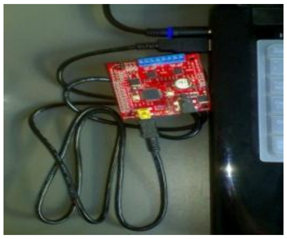
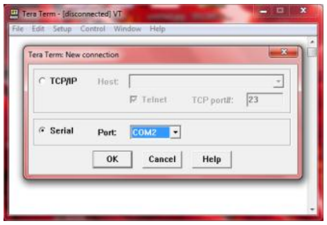


Figure AppIV.1 Cable connection

- Open software and choose serial port which you are using, Press "OK".



Tera Term - [disconnected] VT
File Edit Setup Control Window Help
Tera Term: New connection
OK
Cancel
Help

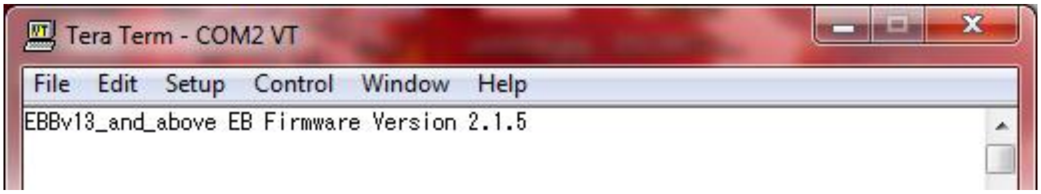
○ TCP/IP Host:

☑ Telnet TCP port#: 23

● Serial Port: COM2

Figure AppIV.2 Port selection

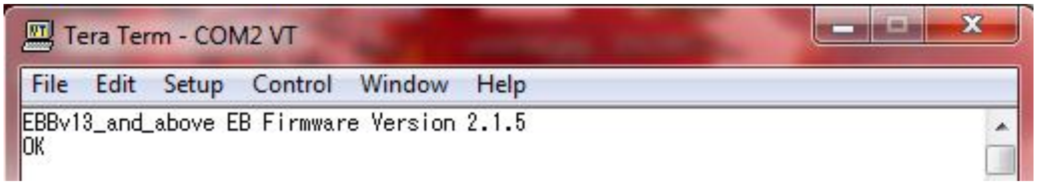
- Send "V" command, and we can see the return packet shown the board version and the firmware version. Figure AppIV.3



```
EBBv13_and_above EB Firmware Version 2.1.5
```

Figure AppIV.3 Board Version Return

- Send “EM, 3” command, and we can see the return packet “OK” as Fig AppIV.4

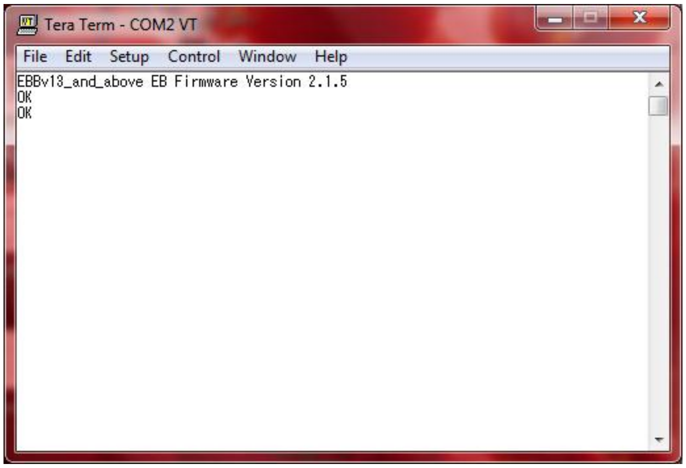


```
EBBv13_and_above EB Firmware Version 2.1.5
OK
```

Figure AppIV.4 ‘EM’ Command Return

- Send “SM,1000,100” command, we also can see that “OK”

Note: this doesn’t mean that the motor must move but it means that this command is sent successful. See Fig AppIV.5.



```
EBBv13_and_above EB Firmware Version 2.1.5
OK
OK
```

Figure AppIV.5 ‘SM’ Command Return

Appendix V

Oscilloscope GUI programing

Although we use the GUI interface to control the system, there is still a m-file existed which support all functions appeared on the GUI front panel and now we are going to introduce how these m-code work.

The functions used for open and close serial port are *fopen* and *fclose* and used *fprintf* for writing data. For example the connection button has the following code:

```
function Connect_Callback(hObject, eventdata, handles)
global PORT
  PORT=serial('com1');
  fopen(PORT);
  set(handles.start_status,'String','Remote');
  fprintf(PORT,':system:dsp "Remote mode start"');
```

The every button has similar code to this example. The port is set to COM1 by a global variable that is easy to callback at all other button. COM1 is the first serial port installed on the computer, but it can be set to any COM port desired with above command. When we use *fopen* function it will be set to its default values: band rate: 9600, data bits: 8, parity: none, stop bits: 1.

We also need send statement *Remote mode start* to the screen of oscilloscope by following command after the port is open.

```
fprintf(PORT,':system:dsp "Remote mode start"');
```

Where the PORT means the port you want to send command to, *:system:dsp* is a command which can display words on the screen and *"Remote mode start"* is the real statement which going to be shown on the screen.

The ON/OFF text indicator is commanded, where needed, with following command

```
Set(handles.start_status,'String','Remote')
Set(handles.start_status,'String','Local')
```

Where *start_status* is the indicator' identifier and *string* is the parameter that is changed and after that is the statement: *Remote* or *Local*. With this command almost every parameter can be changed, like size or position.

For the Time/DIV setting, first we read the value of the slider with the following command:

```
Time=get(handles.Time,'Value')
```

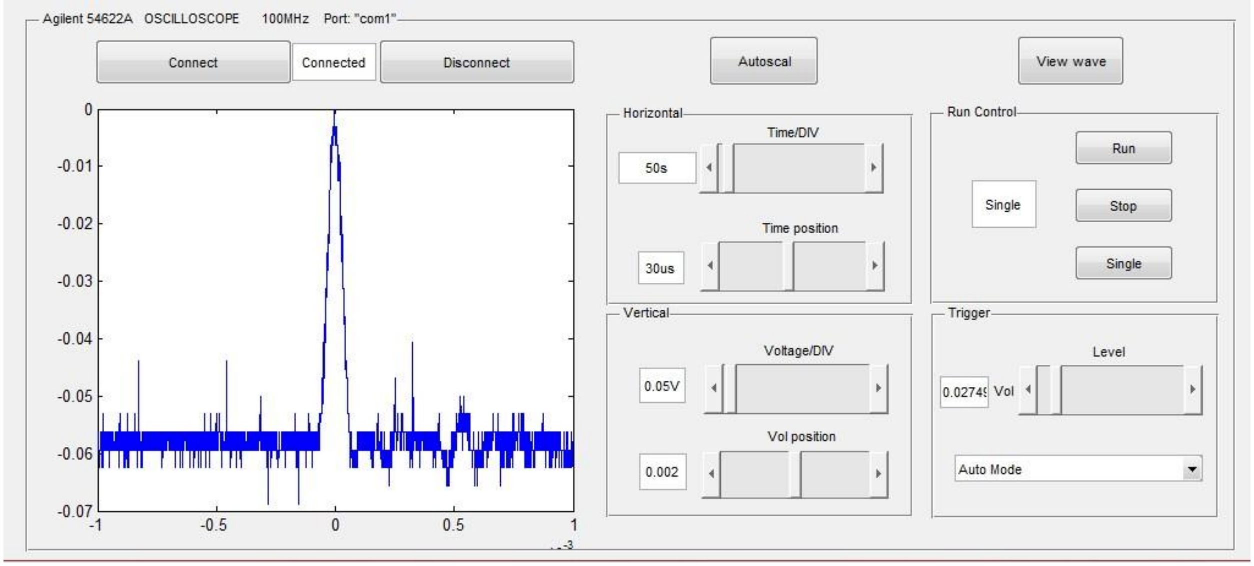


Figure AppV.1 Final Front Panel

Where *Value* is the value read from slider. Then I enumerate each time division which can be set from oscilloscope with following command:

```
if Time==0
    set(handles.Tim,'String','Waitting');
elseif Time>=1&&Time<2
    a=500;
    set(handles.Tim,'String','50s');
elseif Time>=2&&Time<3
    a=200;
    set(handles.Tim,'String','20s');
elseif Time>=3&&Time<4
    a=100;
    set(handles.Tim,'String','10s');
elseif Time>=4&&Time<5
    a=50;
    set(handles.Tim,'String','5s');
   .
   .
   .
elseif Time==31&&Time<32
    a=0.00000005
    set(handles.Tim,'String','5 ns');
else set(handles.Tim,'String','Out of range');
end

Tip=num2str(a);
```

```
kYt=':timebase:range ';
tm=[kYt Tip];
fprintf(PORT,tm);
```

Where *Tip=num2str(a)* is transforming parameter *a* to string, *tm=[kYt Tip]* is a combination of two string *kYt* and *Tip* , and the *fprintf(PORT,tm)* is sending string *tm* to certain port named *PORT.*

The method of setting Time position, Voltage/DIV, Voltage position and Trigger level are all almost same as above Time/DIV setting. The main line is getting value from slider first and then sends it to the oscilloscope. But only different is using related command which already mention at previous USED COMMAND TABLE.

For pop-up menu, the programming is shown as below. It need uses switch to select commands of auto mode or normal mode. As following code said.

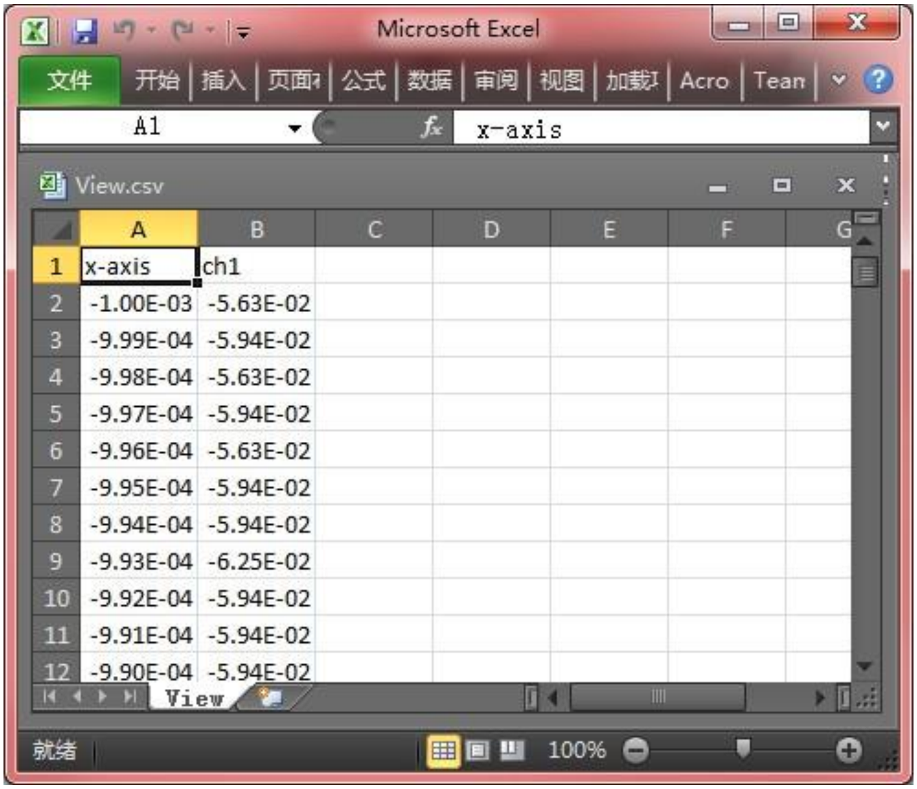
```
tt=get(handles.popupmenu2,'value');
global PORT;
switch tt
    case 1
        fprintf(PORT,':trigger:sweep auto');
    otherwise
        fprintf(PORT,':trigger:sweep normal');
end
```

For waveform reading, we need firstly capture the data of waveform and then plot into the axes into graphical interface as following.

```
aa=csvread('View.csv',1,1,[1,1,2000,1]);
bb=csvread('View.csv',1,0,[1,0,2000,0]);
plot(handles.axes1,bb,aa);
```

Where ‘Wiew.csv’ is a csv file captured by using the software and this procedure is introduced in section 6.4. This file has two columns the one is x-axis another is y-axis. This two columns file format must be selected during capturing data via the software.

Following graph of the CSV file has 2000 sample points also selected during capturing data and the plotted result is shown as figure AppV.1.



	A	B
1	x-axis	ch1
2	-1.00E-03	-5.63E-02
3	-9.99E-04	-5.94E-02
4	-9.98E-04	-5.63E-02
5	-9.97E-04	-5.94E-02
6	-9.96E-04	-5.63E-02
7	-9.95E-04	-5.94E-02
8	-9.94E-04	-5.94E-02
9	-9.93E-04	-6.25E-02
10	-9.92E-04	-5.94E-02
11	-9.91E-04	-5.94E-02
12	-9.90E-04	-5.94E-02

Figure AppV.2

The buttons under Run Control only simply send the commands to oscilloscope by following code.

```
fprintf(PORT,':run')
fprintf(PORT,':stop')
fprintf(PORT,':single');
```

Appendix VI

GUI of Motor controller

The main programming procedure of motor controller is described as below:

Following code indicate how the *Hold up* button works.

```
global PORT
fprintf(PORT,'SM,500,-5000');
```

Where the SM command is already described in the section 5.5, it moves -5000 steps in 500 ms. Although the motor has 200 steps per revolution, due to there is inertia existed we find that the 5000steps moving is almost moving one revolution. And the 500ms is the fastest time that for moving one revolution.

Following code describe how the *Beat* button works.

```
global PORT
fprintf(PORT,'SM,500,5000 ');
```

Where the command 'SM,500,5000' indicates moving hammer 5000 steps in 500 ms.

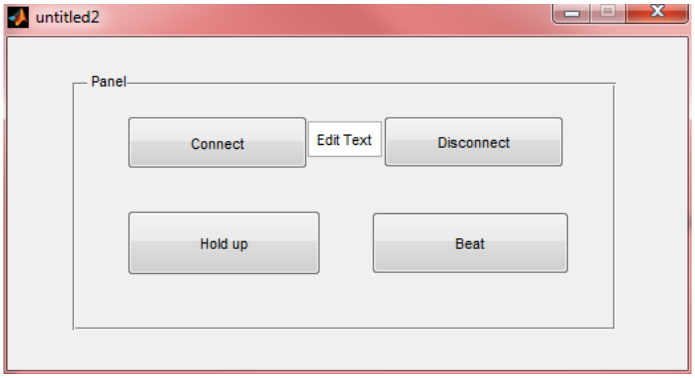


Figure AppVI.1 Final Control Panel

Figure AppVI.1 shows the final complete controlling panel which we can directly control the motor also the hammer without any other procedures.

Appendix VII

Distance Lab Interface Programming Code

Below is the m-file used in the system GUI panel.

```
function varargout = matlabTEST(varargin)
% MATLABTEST MATLAB code for matlabTEST.fig
%      MATLABTEST, by itself, creates a new MATLABTEST or raises the existing
%      singleton*.
%
%      H = MATLABTEST returns the handle to a new MATLABTEST or the handle to
%      the existing singleton*.
%
%      MATLABTEST('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MATLABTEST.M with the given input arguments.
%
%      MATLABTEST('Property','Value',...) creates a new MATLABTEST or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before matlabTEST_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to matlabTEST_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help matlabTEST

% Last Modified by GUIDE v2.5 09-May-2012 14:58:32

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @matlabTEST_OpeningFcn, ...
                   'gui_OutputFcn',  @matlabTEST_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before matlabTEST is made visible.
function matlabTEST_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to matlabTEST (see VARARGIN)

% Choose default command line output for matlabTEST
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);



% UIWAIT makes matlabTEST wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = matlabTEST_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;




function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in Connect.
function Connect_Callback(hObject, eventdata, handles)
% hObject    handle to Connect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global PORT
set(handles.ONOFF,'String','Load...');
PORT=serial('com1');
fopen(PORT);
fprintf(PORT,':system:dsp "Remote mode being start"');
set(handles.ONOFF,'String','On');



% --- Executes on button press in Disconnect.
function Disconnect_Callback(hObject, eventdata, handles)
% hObject    handle to Disconnect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global PORT
fprintf(PORT,':system:dsp " Disconnected !!"');
fclose(PORT);
set(handles.ONOFF,'String','Off');



% --- Executes on button press in Autoscal.
function Autoscal_Callback(hObject, eventdata, handles)
% hObject    handle to Autoscal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%ss=serial('com1');
%fopen(ss);
global PORT;
fprintf(PORT,':autoscale');
%fclose(ss);


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Oq=csvread('View.csv',1,1,[1,1,2000,1]);
Ox=csvread('View.csv',1,0,[1,0,2000,0]);

plot(handles.axes1,Ox,Oq);






function Y_P_Callback(hObject, eventdata, handles)
% hObject    handle to Y_P (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Y_P as text
%        str2double(get(hObject,'String')) returns contents of Y_P as a double


% --- Executes during object creation, after setting all properties.
function Y_P_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Y_P (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Vol_Callback(hObject, eventdata, handles)
% hObject    handle to Vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Vol as text
%        str2double(get(hObject,'String')) returns contents of Vol as a double


% --- Executes during object creation, after setting all properties.
function Vol_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Vol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function slider9_Callback(hObject, eventdata, handles)
% hObject    handle to slider9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
A=get(handles.slider9,'Value')
Y=A/0.4*0.05
Yy=num2str(Y)
yy=[Yy 'V']
set(handles.Vol,'String',yy);
```

```
str=num2str(A);
k=':channel1:range ';
y=[k str];
global PORT
%b=serial('com1');
%fopen(b);
fprintf(PORT,y);
%fclose(b);


% --- Executes during object creation, after setting all properties.
function slider9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



% --- Executes on slider movement.
function YPosition_Callback(hObject, eventdata, handles)
% hObject    handle to YPosition (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
Yt=get(handles.YPosition,'Value')
set(handles.Y_P,'String',num2str(Yt));
stY=num2str(Yt);
kY=':channel1:offset ';
yY=[kY stY];
global PORT
%b=serial('com1');
%fopen(b);
fprintf(PORT,yY);

% --- Executes during object creation, after setting all properties.
function YPosition_CreateFcn(hObject, eventdata, handles)
% hObject    handle to YPosition (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function Time_Callback(hObject, eventdata, handles)
% hObject    handle to Time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine rangeof slider
global PORT;
Time=get(handles.Time,'Value')
if Time==0
    set(handles.Tim,'String','Waitting');
elseif Time>=1&&Time<2
    a=500;
    set(handles.Tim,'String','50s');
elseif Time>=2&&Time<3
       a=200;
       set(handles.Tim,'String','20s');
elseif Time>=3&&Time<4
       a=100;
       set(handles.Tim,'String','10s');
elseif Time>=4&&Time<5
       a=50;
       set(handles.Tim,'String','5s');
elseif Time>=5&&Time<6
    a=20;
    set(handles.Tim,'String','2s');
elseif Time>=6&&Time<7
    a=10;
    set(handles.Tim,'String','1s');
elseif Time>=7&&Time<8
    a=5;
    set(handles.Tim,'String','500 ms');
elseif Time>=8&&Time<9
    a=2;
    set(handles.Tim,'String','200 ms');
elseif Time>=9&&Time<10
    a=1;
    set(handles.Tim,'String','100 ms');
elseif Time>=10&&Time<11
    a=0.5;
    set(handles.Tim,'String','50 ms');
elseif Time>=11&&Time<12
    a=0.2;
    set(handles.Tim,'String','20 ms');
elseif Time>=12&&Time<13
    a=0.1;
    set(handles.Tim,'String','10 ms');
elseif Time>=13&&Time<14
    a=0.05;
    set(handles.Tim,'String','5 ms');
elseif Time>=14&&Time<15
    a=0.02;
    set(handles.Tim,'String','2 ms');
elseif Time>=15&&Time<16
    a=0.01;
    set(handles.Tim,'String','1 ms');
elseif Time>=16&&Time<17
    a=0.005;
    set(handles.Tim,'String','500 µs');
elseif Time>=17&&Time<18
    a=0.002;
    set(handles.Tim,'String','200 µs');
elseif Time>=18&&Time<19
    a=0.001;
    set(handles.Tim,'String','100 µs');
elseif Time>=19&&Time<20
    a=0.0005;
    set(handles.Tim,'String','50 µs');
elseif Time>=20&&Time<21
    a=0.0002;
    set(handles.Tim,'String','20 µs');
elseif Time>=21&&Time<22
    a=0.0001;
    set(handles.Tim,'String','10 µs');
elseif Time>=22&&Time<23
    a=0.00005;
    set(handles.Tim,'String','5 µs');
elseif Time>=23&&Time<24
    a=0.00002;
    set(handles.Tim,'String','2 µs');
elseif Time>=24&&Time<25
    a=0.00001;
    set(handles.Tim,'String','1 µs');
elseif Time>=25&&Time<26
    a=0.000005;
    set(handles.Tim,'String','500 ns');
elseif Time>=26&&Time<27
    a=0.000002;
    set(handles.Tim,'String','200 ns');
elseif Time>=27&&Time<28
    a=0.000001;
    set(handles.Tim,'String','100 ns');
elseif Time>=28&&Time<29
    a=0.0000005
    set(handles.Tim,'String','50 ns');
elseif Time>=29&&Time<30
    a=0.0000002
    set(handles.Tim,'String','20 ns');
elseif Time==30&&Time<31
    a=0.0000001
    set(handles.Tim,'String','10 ns');
elseif Time==31&&Time<32
    a=0.00000005
    set(handles.Tim,'String','5 ns');
else set(handles.Tim,'String','Out of range');
end

Tip=num2str(a);
kYt=':timebase:range ';
tm=[kYt Tip];
fprintf(PORT,tm);
```

```
% --- Executes during object creation, after setting all properties.
function Time_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function Tim_Callback(hObject, eventdata, handles)
% hObject    handle to Tim (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Tim as text
%        str2double(get(hObject,'String')) returns contents of Tim as a double


% --- Executes during object creation, after setting all properties.
function Tim_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Tim (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function Trigger_Callback(hObject, eventdata, handles)
% hObject    handle to Trigger (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
t=get(handles.Trigger,'Value')
set(handles.edit6,'String',num2str(t));
st=num2str(t);
kt=':trigger:level ';
yt=[kt st];
global PORT
%b=serial('com1');
%fopen(b);
fprintf(PORT,yt);
%fclose(b);

% --- Executes during object creation, after setting all properties.
function Trigger_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to Trigger (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu2
tt=get(handles.popupmenu2,'value');
global PORT;
switch tt
    case 1
        fprintf(PORT,':trigger:sweep auto');
    otherwise
        fprintf(PORT,':trigger:sweep normal');
end

% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function slider14_Callback(hObject, eventdata, handles)
% hObject    handle to slider14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
B=get(handles.slider14,'Value');
Y=-B/0.4*10
Yy=num2str(Y)
yy=[Yy 'us']
set(handles.TimeP,'String',yy);

k='timebase:delay ';
y=[k yy]

global PORT
%b=serial('com1');
%fopen(b);
fprintf(PORT,y);

% --- Executes during object creation, after setting all properties.
function slider14_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function TimeP_Callback(hObject, eventdata, handles)
% hObject    handle to TimeP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of TimeP as text
%        str2double(get(hObject,'String')) returns contents of TimeP as a double


% --- Executes during object creation, after setting all properties.
function TimeP_CreateFcn(hObject, eventdata, handles)
% hObject    handle to TimeP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Run.
function Run_Callback(hObject, eventdata, handles)
% hObject    handle to Run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global PORT;
fprintf(PORT,':run')
set(handles.Status,'String','Run');


% --- Executes on button press in Stop.
function Stop_Callback(hObject, eventdata, handles)
% hObject    handle to Stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global PORT;
fprintf(PORT,':stop')
set(handles.Status,'String','Stop');


% --- Executes on button press in Single.
function Single_Callback(hObject, eventdata, handles)
% hObject    handle to Single (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global PORT;
fprintf(PORT,':single');
set(handles.Status,'String','Single');



function Status_Callback(hObject, eventdata, handles)
% hObject    handle to Status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Status as text
%        str2double(get(hObject,'String')) returns contents of Status as a double


% --- Executes during object creation, after setting all properties.
function Status_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Motor_hold.
function Motor_hold_Callback(hObject, eventdata, handles)
% hObject    handle to Motor_hold (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global PORT_M
fprintf(PORT_M,'SM,1000,2000');
set(handles.Motor_up_dowm,'String','Go up');

% --- Executes on button press in Motor_heat.
function Motor_heat_Callback(hObject, eventdata, handles)
% hObject    handle to Motor_heat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global PORT_M
fprintf(PORT_M,'EM,1');
fprintf(PORT_M,'SM,1000,-10000');

set(handles.Motor_up_dowm,'String','Go down');


% --- Executes on button press in Motor_connection.
function Motor_connection_Callback(hObject, eventdata, handles)
% hObject    handle to Motor_connection (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global PORT_M
PORT_M=serial('com2');
fopen(PORT_M);
fprintf(PORT_M,'SM,1000,1000');
set(handles.Motor_Con_status,'String','On');


% --- Executes on button press in Motor_disconnection.
function Motor_disconnection_Callback(hObject, eventdata, handles)
% hObject    handle to Motor_disconnection (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global PORT_M
fclose(PORT_M);
set(handles.Motor_Con_status,'String','Off');
set(handles.Motor_up_dowm,'String','Status');

function Motor_Con_status_Callback(hObject, eventdata, handles)
% hObject    handle to Motor_Con_status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Motor_Con_status as text
%        str2double(get(hObject,'String')) returns contents of Motor_Con_status as a double
% --- Executes during object creation, after setting all properties.
function Motor_Con_status_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Motor_Con_status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Motor_up_dowm_Callback(hObject, eventdata, handles)
% hObject    handle to Motor_up_dowm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Motor_up_dowm as text
%        str2double(get(hObject,'String')) returns contents of Motor_up_dowm as a double


% --- Executes during object creation, after setting all properties.
function Motor_up_dowm_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Motor_up_dowm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function ONOFF_Callback(hObject, eventdata, handles)
% hObject    handle to ONOFF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ONOFF as text
%        str2double(get(hObject,'String')) returns contents of ONOFF as a double


% --- Executes during object creation, after setting all properties.
function ONOFF_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ONOFF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end
```