An Ultrasonic Sensing and Indentation Apparatus for Assessment of Tissue Geometry and Mechanical Properties

by

Zixi Liu

B.S., Massachusetts Institute of Technology (2016)

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Master Of Engineering in Electrical Engineering and Computer Science at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2017 [September 2017]

© Massachusetts Institute of Technology 2017. All rights reserved.

Signature redacted

Author ............... ............... 

Department of Electrical Engineering and Computer Science

Signature redacted

August 18, 2017

Certified by....... ............... 

Hugh M. Herr

Professor of Media Arts and Sciences

Thesis Supervisor

Signature redacted

Accepted by ............... ............... 

Christopher J. Terman

Chairman, Master of Engineering Thesis Committee
An Ultrasonic Sensing and Indentation Apparatus for 
Assessment of Tissue Geometry and Mechanical Properties 

by 

Zixi Liu 

Submitted to the Department of Electrical Engineering and Computer Science 
on August 18, 2017, in partial fulfillment of the 
requirements for the degree of 
Master Of Engineering in Electrical Engineering and Computer Science 

Abstract 

Measuring the distance from the skin to the bone and soft tissue mechanical properties is important to custom designing prosthetic sockets for amputee patients using a computer aided method. The current state-of-the-art method to obtain such information is via MRI scans. However, MRI scans are expensive, not widely accessible, and may not be as accurate if there is a time gap between when the MRI scan is taken and when the design process takes place. In this thesis, I designed and implemented a hand-held apparatus which measures both the skin-to-bone depth and soft tissue mechanical properties. With a PC interface, this method involves gathering and processing data from an ultrasound transducer, a force sensor, and an accelerometer. The procedure of use involves rotating the apparatus around the limb while maintaining a light contact to acquire skin-to-bone depth, and indenting the apparatus into the limb to acquire soft tissue mechanical properties. Here I show that a miniaturized apparatus as such can measure tissue boundaries and tissue indentation with sub-millimeter precision and out performs a commercial ultrasound imaging system in my case study, which makes custom computer prosthetic socket design easier, more affordable, and more accessible. 

Thesis Supervisor: Hugh M. Herr 
Title: Professor of Media Arts and Sciences
Acknowledgments

It has been an incredible journey working on this project. Along this journey, I was lucky to have some truly amazing people guiding and supporting me, without whom this work would not have been possible.

First of all, I would like to thank my advisor Professor Hugh Herr for giving me the opportunity to pursue this project at the Biomechatronics Group (Biomech), supporting me along my journey, making an impact on my career, and inspiring me for a lifetime. I must also thank my supervisor Kevin Moerman, for his guidance, support, and mentorship, without which I could not have grown as much as a researcher and scientist.

I would also like to extend my gratitude to my academic advisor Professor Joel Voldman, who mentored me both as an undergraduate student since my sophomore year and as a graduate student, for his support, mentorship, and patience. I truly appreciate his wisdom and guidance to my search of career passion and life goals.

I must also thank my colleagues Steven Keyes, who introduced me to this project, met with me regularly, and provided valuable feedback; Dana Solav, who kindly helped me with setting up, running, and learning about the DIC experiments. I am also grateful to all my other colleagues at Biomech, who not only helped me with various details about my project, but also provided valuable advice for my career. In particular, I would like to acknowledge Anthony Zorzos, Bryan Ranger, Matt Carney, Jessica Beltran Ullari, and Tyler Clites. I would also like to thank my office mates Jiun-Yih Kuan and David Hill, for the good times we shared, and the encouragement towards pursuing my future.

I would like to thank Robin Singh and Shriya Srinivasan for kindly sharing lab equipment with me, making it all possible. I am also thankful to my lab administrative manager Lindey Reynolds, who provided support throughout my time here at Biomech, making every non-technical aspect incredibly simple.

I am also thankful to my family for their unconditional love and support, without whom, I could never have walked this far. In particular, I would like to thank my
parents who inspired me to become who I am and to keep exploring who I want to be, and supported me no matter what. I would also like to thank Kathy and Jack Barry for always being there for me and for caring about me and supporting me.

Last but not least, I would like to thank all my dear friends, without whom I would not have been where I am today: Aashish, Caihang, Chandan, Dana, Hilena, Ishwar, Iveel, Kishore, Pashu, Sapna, Sarah, Surya, Yi, and all other friends at MIT Student House.
# Contents

1 Introduction 15

1.1 Ultrasonic Sensing for Tissue Boundaries 16

1.2 Force and Ultrasonic Sensing for Mechanical Properties of Soft Tissues 17

1.3 Outline of the Thesis 19

2 Theoretical Background 21

2.1 Ultrasonics Fundamentals 21

2.1.1 Piezoelectric Elements 21

2.1.2 Sound in the Human Body 22

2.2 Transducer Selection Considerations 23

2.2.1 Transducer Frequency 23

2.2.2 Transducer Diameter 25

2.2.3 Transducer Categories 26

2.3 Driving and Receiving 27

3 System Design 29

3.1 Overall System Structure 29

3.2 Transducer Selection 31

3.2.1 Transducer in Probe Assembly 31

3.2.2 Further Transducer Selection 31

3.3 Design of Probe Assembly 32

3.4 Use Procedures 34

3.4.1 Tissue Boundaries Detection Procedure 34
3.4.2 Indentation Experiment Procedure .................................. 34
3.5 Data Logging ................................................................. 35
3.5.1 Visual Feedback Mode .................................................... 36
3.5.2 Blind Mode ................................................................. 39

4 Signal Processing ............................................................... 41
4.1 Ultrasound ................................................................. 41
  4.1.1 General Waveform Processing .............................................. 42
  4.1.2 Processing Method For Indentation Experiments ................ 43
  4.1.3 A Histogram and Edge Based Method for Tissue Boundaries Detection ......................................................... 43
4.2 Self-Weight Compensation ............................................... 45
  4.2.1 Calibration ............................................................... 46
  4.2.2 Self-Weight Compensation .............................................. 46

5 Experiments and Verification ............................................... 49
5.1 Depth Sensing Verification by a Phantom Staircase .............. 49
5.2 Indentation Verification by DIC in Phantom ..................... 53
5.3 Indentation Verification by DIC in-Vivo ......................... 55
5.4 A Comparison with a Commercial Ultrasound System with Respect to “Ground Truth” by MRI in-Vivo ......................... 57
5.5 Error Analysis ............................................................... 62
  5.5.1 Error in the Phantom ................................................... 62
  5.5.2 Error in-Vivo ........................................................... 62

6 Discussion ................................................................. 65
6.1 Cost Breakdown .......................................................... 68
6.2 Future Design Recommendations ..................................... 68

A Code ................................................................................. 71
A.1 Data Logging .............................................................. 71
  A.1.1 Blind Mode Depth Sensing ........................................... 71
A.1.2 Blind Mode Indentation Sensing ....................................... 77
A.1.3 Visual Feed Back Mode In Phantom ................................. 85
A.1.4 Visual Feed Back Mode in-Vivo ..................................... 96

A.2 Post Processing .............................................................. 107
A.2.1 Post Processing of Depth Sensing in Phantom ...................... 107
A.2.2 Post Processing of Depth Sensing in-Vivo ......................... 111
A.2.3 Post Processing of Indentation Sensing in Phantom ............... 116
A.2.4 Post Processing of Indentation Sensing in-Vivo .................. 120

A.3 MRI Image Processing ...................................................... 124
A.3.1 Import DICOM Data .................................................. 124
A.3.2 View Converted DICOM Data ....................................... 128

A.4 DIC ................................................................................. 130
A.4.1 Compare Results, in Phantom ....................................... 130
A.4.2 Compare Results, in-Vivo ............................................. 133
A.4.3 Find Indentation ......................................................... 136
A.4.4 Process Raw Images ................................................... 139

A.5 Other ................................................................................. 167
A.5.1 Plotting support ......................................................... 167
A.5.2 Staircase Calibration ..................................................... 167
# List of Figures

2-1 A side view of a piezoelectric ultrasonic transducer as shown in patent[3] 26

3-1 System Diagram ......................................................... 30
3-2 The Probe Assembly .................................................... 33
3-3 Using the Probe for Tissue Boundaries Detection .................. 35
3-4 Using the Probe for Indentation ...................................... 35
3-5 Visual Feedback Mode Diagram ....................................... 37
3-6 Visual Feedback Mode Display Example .............................. 38
3-7 Blind Mode Diagram .................................................... 39

4-1 An Example Trail Processed with the Histogram and Edge Based Method 44
4-2 Tilt angles from a tri-axis accelerometer as in [24] .................. 47
4-3 Accelerometer Directions as in the Probe .............................. 48

5-1 Phantom in a Staircase .................................................. 50
5-2 Preliminary Staircase Experiment for Speed of Sound ............ 51
5-3 Phantom Staircase Depth Detection .................................... 52
5-4 Camera Views of DIC Experiments in Phantom .................... 53
5-5 Phantom Staircase Indentation Experiment Error Distribution .... 54
5-6 Example Data of Depth, Force, and Acceleration Reading in an Indentation Experiment .................................................. 55
5-7 Camera Views of DIC Experiments in-vivo ............................ 56
5-8 In-vivo Indentation Experiment Error Distribution ................ 56
5-9 Paths to Tibia and Fibula from Markers ............................... 58
<table>
<thead>
<tr>
<th>Page</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-10</td>
<td>Measurement vs. MRI Results and Commercial Ultrasound Results</td>
<td>60</td>
</tr>
<tr>
<td>5-11</td>
<td>Error Histogram of Case Study</td>
<td>61</td>
</tr>
<tr>
<td>6-1</td>
<td>A Hand-Held Apparatus</td>
<td>65</td>
</tr>
<tr>
<td>6-2</td>
<td>Estimated PDF from Four Experiments</td>
<td>66</td>
</tr>
<tr>
<td>6-3</td>
<td>Two Sets of Four Experiments and Results</td>
<td>67</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Component Names and Description .................................................. 30
3.2 Transducer Characteristics .............................................................. 31
3.3 A Possible Transducer for Near-skin Tissue Boundaries Detection .... 32

4.1 Accelerometer Raw ............................................................................ 46

5.1 Phantom Staircase Depth Detection Experiment Result ................. 51
5.2 Phantom Staircase Indentation Experiment Result .......................... 54
5.3 In-vivo Indentation Experiment Result .............................................. 56
5.4 Distances from Markers to Tibia and Fibula from Markers .......... 57
5.5 Mean and Standard Deviation of Error Compared to MRI Results .. 59

6.1 Cost Breakdown of the System after Further Miniaturization ......... 68
Chapter 1

Introduction

For lower limb amputees, a comfortable prosthetic socket is critical. Improperly fit prosthetic sockets can cause soft tissue damage such as ulcers and blisters [14], as well as deep tissue injury [18]. At present, the design and manufacture of prosthetic socket is still largely artisanal. As a result, the end product is non-repeatable and varies from prosthetist to prosthetist. To make it more data driven, repeatable, and predictable, modern approaches employ computer aided methods for strain evaluation [18], prosthetic socket design, and manufacturing [21]. The state-of-the-art prosthetic socket design uses limb geometry, tissue boundaries\(^1\), and soft tissue non-linear elastic and viscoelastic mechanical behavior as design parameters. These parameter values are acquired using magnetic resonance imaging (MRI), and inverse finite element analysis (FEA) of in-vivo indentation experiments [21]. However, MRI is expensive and not world-wide accessible. Additionally, the necessity of up-to-date MRI scans further increases the challenge.

In an attempt to make computer aided socket techniques more widely accessible, this thesis describes an alternative method to measure tissue boundaries and soft tissue mechanical behavior using ultrasonic and force sensing. Ultrasonic and force sensing are two of the most affordable and safe options, and are both largely independent of location (altitudes, humidity, temperature, etc), producing repeatable and

\(^1\)In this thesis, the term tissue boundaries refer to both soft tissue boundaries, such as muscle to fat, and soft tissue to bone boundary.
reliable results. In particular, given that force sensing is a mature, autonomous technology\footnote{Force sensors such as force sensing capacitors, resistors, strain gauges, and load cells are, once mounted, independent of user operation, producing repeatable results as long as the mounting or system design does not introduce instability.}, this thesis focuses on the procedure and processing of the ultrasonic sensing. The objectives of this project are to make tissue boundary and mechanical properties sensing more affordable, accessible, and portable. In discussion of the feasibility of this method, this thesis intensively discusses the design, implementation, analysis, and verification of a particular prototype.

1.1 Ultrasonic Sensing for Tissue Boundaries

This thesis chooses to explore the feasibility of using ultrasonic sensing for tissue boundaries detection, in particular bone location detection, among other available technologies such as CT scans and X-rays because ultrasound is non-radiative, the most affordable, and can be non-invasive. So far ultrasonic technologies for depth detection in-vivo mainly involves either imaging using a probe of multiple ultrasonic crystals (crystals) or invasive techniques. Therefore, this thesis attempts to fill the gap with a non-invasive technique with a single crystal.

Widespread for both industrial and medical use, ultrasonic technology is a mature field where sensors of a variety of types specifically designed for various applications are commonly available. While medical ultrasound machines are already significantly more affordable and accessible than MRI scanners, individual ultrasonic sensors (ultrasonic transducers) are even more portable and affordable.

In previous work, there are mainly two approaches for in-vivo distance sensing using ultrasonic technologies. The most common one uses hospital standard ultrasound machines. Ultrasonic measurements are routinely performed in hospitals not only as part of diagnostic assessment such as of various organs, tumors, and the fetus during pregnancy [20], but also as a guidance in surgeries [2]. Commercial ultrasound machines uses a probe of arrays of ultrasonic transducers to produce images based on the reflected signals at tissue boundaries. One can use image segmentation and
tissue characterization to determine tissue boundaries [17].

Another common method is sonomicrometry. Sonomicrometry is a particular technique among ultrasonic technologies that measures the distance between piezoelectric elements (crystals\(^3\)). In literature, sonomicrometry is also considered a reliable method for *in-vivo* measurements [14]. The main difference between sonomicrometry and industrial ultrasonic probes and transducers is that while sonomicrometry requires at least two crystals (one transmitter and one receiver), and assumes that the ultrasonic wave will travel through the medium and reach the receiver, other ultrasonic probes and transducers tend to measure the echo from substances in the medium. For instance, hospital standard ultrasound imaging machines measure the echoed ultrasonic waves bounced back at tissue boundaries. In an *in-vivo* setting, sonomicrometry casts additional constraint on the positions and stability of the crystal pairs. In practice, the crystals are implanted to the human organs or other body parts of measurement, which is a more invasive method compared to the imaging techniques [14]. Regardless, both use the same acoustic principles in calculating the distance of measurement.

In this thesis, we examine the feasibility of using echo-based ultrasonic sensing with a single-element ultrasonic transducer to determine the tissue boundaries based on a set of single-dimensional measurements.

### 1.2 Force and Ultrasonic Sensing for Mechanical Properties of Soft Tissues

Adding force sensing, the same apparatus can serve both roles of a tissue boundaries detector and an indentation experiment tool. Indentation experiment support is included in the system design requirement because they are a crucial part of an inverse FEA method, which best captures the mechanical properties of the limb. In previous

\(^3\)Piezoelectric elements and crystals both refer to the fundamental parts of ultrasonic transducers. In this thesis, the terms piezoelectric elements, crystals, ultrasonic transducers, ultrasonic crystals, and transducers are used interchangeably as referring to a single ultrasonic sensing unit.

17
work using an inverse FEA method with indentation experiments, measurements are taken to obtain a force-displacement history. With force sensing by force sensing and displacement sensing by ultrasonic sensing, force-displacement history can easily be acquired by the apparatus for indentation experiments.

FEA methods are suitable for modeling the mechanical properties of the limb because, compared to rigid bodies, soft tissue observes a more dramatic shape change, distribution of force, and recovery curve under pressure. Previous work adapts a FEA approach for modeling soft tissue. During FEA, the non-linear elastic behavior of soft tissue is modeled using hyper-elastic formulations, and viscoelasticity is modeled using the quasi-linear theory of viscoelasticity. Having measurements of distinctively representative anatomical regions, an inverse FEA based optimization routine determines the material parameters for each soft tissue location of measurement using the force-time and force-displacement curves of indentation. Since these parameters are descriptive of anatomically distinctly representative locations, one can then model the entire limb of measurement with the acquired parameters [21]. Although in [21], a robotically actuated system were used for indentation experiment, with the knowledge of the trajectory of the indenter tip, an unconstrained indentation experiment using apparatus as described in this thesis is sufficient to determine the modeling parameters in an inverse FEA method.

In this thesis, force is measured with a capacitive force sensor through a microcontroller. Due to nonlinearity introduced in apparatus prototype design, validation for using ultrasound and force sensing for indentation experiment is out of the scope of this thesis although the concept and the preliminary results (with inaccurate force reading) are shown in chapter 5.

---

4 One can obtain the trajectory via methods such as digital image correlation (DIC) as shown in verification example in later chapters.

5 Due to time constrain, the sensor was selected because it was available at the lab despite better options such as load cells may improve the results.
1.3 Outline of the Thesis

Theoretical Background
This chapter highlights the fundamentals of ultrasonic sensing, transducer selection considerations such as transducer frequency, dimensions, and transducer type, and the driving and receiving of an ultrasonic signal.

System Design
This chapter describes the system design from the following four aspects: overall system structure, the transducer of selection, a probe assembly and its procedure of use, and two modes of user interface. First, the system structure of sensors, data acquisition, and communication are described. Following the overall system structure, the analysis of the ultrasonic transducer of selection is discussed. Then, a hand-held probe assembly apparatus that centralizes all sensors and the microcontroller is described with design details and use procedures for tissue boundaries detection and indentation experiment. Finally, this chapter also describes the two modes of data logging, one for familiarizing the user with ultrasonic transducer measurements, and one for more efficient data acquisition.

Signal Processing
This chapter describes the processing of ultrasonic signal, and acceleration signal for self-weight compensation. General ultrasound waveform processing is described followed by additional rules for indentation sensing and a histogram and edge based method for tissue boundaries detection with a single-element ultrasonic transducer.

Experiments and Verification
To verify that the system design meets the objectives of an accurate tissue boundary and indentation detection using low profile ultrasonic transducers, in-vivo experiments are performed and results are compared to “ground truth” from MRI scans and DIC results, respectively. To understand the limits of ultrasonic sensing for this application, the tissue boundary detection results are also compared to that from a commercial ultrasound imaging system. Furthermore, a phantom staircase study for
both depth and indentation are performed to remove the inaccuracy introduced by human factors in \textit{in-vivo} experiments.

In this chapter, two sets of four experiments are described: depths and indentation experiments in a phantom staircase, and bone depths and indentation experiments \textit{in-vivo}. Following the experiment methods and results, this chapter discusses sources of errors in the system, for phantom gel and for human tissue.

\textit{Discussion}

This chapter recapitulates the problem, the solutions, and the results. A cost breakdown of the system and recommendations for future improvements are also discussed.
Chapter 2

Theoretical Background

This chapter highlights the fundamentals of ultrasonic sensing, transducer selection considerations such as transducer frequency, dimensions, and transducer type, and the driving and receiving of an ultrasonic signal.

2.1 Ultrasonics Fundamentals

Time-based ultrasonic technologies measures the inside of human body in a noninvasive manner by measuring the time lapse between the emitting of the sound wave and the receiving of the echo with the transducer in contact with the skin. This section discusses the fundamentals of ultrasonic transducers.

2.1.1 Piezoelectric Elements

Ultrasonics typically refers to piezoelectric transducers or capacitive ultrasonic transducers. While the latter is a relatively new concept in the field of ultrasonic sensing [1], the former is a mature technology commonly used in industrial Non-Destructive Testing (NDT) [6] and medical imaging. This thesis describes an apparatus that uses a single piezoelectric ultrasound transducer, which is typical for industrial NDT but not so much for medical use.

Piezoelectrics are materials that can be polarized, or responds to, both an electric
field or a mechanical stress. Polarization is when the medium adapts to perturbation by dynamically changing the positions of the nuclei and the electrons. This polarization generates an electric field that can be used to transform the mechanical energy, used in the material's deformation, into electrical energy, or vice versa [4]. This means that when an ultrasonic wave hits the piezoelectric material, the mechanical energy in the ultrasonic wave causes deformation of the piezoelectric material, which generates an electric field in the piezoelectric material, such that the piezoelectric material generates an electrical signal upon hit by an ultrasonic wave. In addition, when the piezoelectric material is shocked with large electric voltage (on the order of magnitude of 10 to 100V in the particular application of tissue boundaries detection), this electrical shock can cause the piezoelectric material to change its shape, causing a mechanical vibration, that generates an ultrasonic wave, a process called the reverse piezoelectric effect [5]. This property of piezoelectric material also allows it to serve the role of transmitter and receiver indifferently. In fact, some transducers(such as the one used in this thesis) are designed as a single element transducer such that the same piezoelectric material will transmit and receive the transmitted ultrasonic signal.

For detailed theory on piezoelectric phenomenon, one may refer to standard literature on piezoelectricity (Mason in Electromechanical Transducers and Wave Filters, 1942; Mason in Physical Acoustics and the Properties of Solids, 1958; Cady in Piezoelectricity, 1946; Nalwa in Ferroelectric Polymers: Chemistry, Physics, and Applications, 1995; Ikeda in Fundamentals of Piezoelectricity, 1996) [4]. For detailed introduction on ultrasound physics and transducers, one may refer to introductory literatures (Edelman in Understanding Ultrasound Physics, 2012, etc.) and product brochures from ultrasonic transducer companies.

2.1.2 Sound in the Human Body

As sound waves propagates through tissue boundaries, such as from fat to muscle, muscle to bone, some are reflected while others are refracted. Fortunately the perturbation to ultrasound waves on propagation through soft tissue is sufficiently small
that refraction and defocussing artifacts can often be neglected [8]. Situations where these effects may become significant often takes place through fatty tissues, such as the breast [10], or through the skin/fat/muscle complex of the abdominal surface [9].

This thesis aims to assist prosthetic socket technology, thus specifically narrows the usage to residual limbs where neither fatty tissue nor skin/fat/muscle complex is likely to be significant. With the assumption that most soft tissues in the residual limb observes similar acoustic properties, this thesis models the limb with soft tissue of fixed speed of sound and bones only; further implication to this assumption is discussed in Chapter 5.

The reflected sound waves are the receiver signals to the ultrasonic transducers. The sound signal attenuates as it propagates through the human body. The attenuation is a function of the distance of traveling and the frequency of the sound wave: the further the sound travels, the greater the attenuation, the shape of which depends on the transducer but most captures a similar-to-exponential-decay form [5]; the attenuation in soft tissues increases approximately linearly as frequency increases [13].

2.2 Transducer Selection Considerations

Design considerations are optimized for the specific requirements. The most important trade-offs in meeting a specific design requirement include axial resolution, focus area, and the dimension of the transducer in the context of tissue boundary detection using a single element transducer. Transducer frequency, diameter, and transducer type are the determining factors and thus the most common design considerations.

2.2.1 Transducer Frequency

Ultrasound is defined by sound of higher than 20kHz frequency. Typical medical ultrasonic imaging uses frequency range from 2 to 15MHz [11]. The speed of sound is 1450 m/s in fat, 1550-1630 m/s in muscle [13], and 1540 m/s in soft tissue [22]. In ultrasound beam-forming, calculation typically assume a fixed sound speed (e.g. 1540
This thesis assumes 1540 m/s speed of sound in the limb of measurement. Further discussion on what error this assumption might bring is made in Chapter 5.

The wavelength of the ultrasonic wave is given by

$$\lambda = \frac{c}{f}$$

(2.1)

where $\lambda$ is the wavelength, $c$ is the speed of sound, and $f$ is the frequency of the ultrasonic wave. From equation 2.1, ultrasonic waves of 1 to 15 MHz have 1.54 to 0.10 mm wavelength.

The wavelength determines the axial resolution of the ultrasonic transducer. Axial resolution is one of the five resolutions that ultrasonics concern: longitudinal, axial, range, radial, and depth (LARRD). Axial resolution describes how finely the signal can tell apart two structures that are parallel to the sound beam's main axis, i.e. if there are two structures aligned with the direction of the sound beam, how far apart do they have to be for the transducer to differentiate the two. Axial resolution is particularly important for a single-element transducer since it receives a one dimensional signal.

Axial resolution is not only a function of wavelength but also that of the number of cycles in the triggering electrical pulse. Depending on the driver system that the application adapts, pulses of different shapes, duration, and cycles may differ. Square waves, sine waves, or waves in between of one to ten cycles are the most common in commercial driver systems. The axial resolution is given by

$$r = \frac{\lambda \times n}{2}$$

(2.2)

where $r$ is the axial resolution (mm), $\lambda$ is the wavelength of the sound wave (mm), and $n$ is the number of cycles in pulse. In the case of an 1MHz transducer driven by one cycle per pulse, the axial resolution is 0.77 mm. The resolution becomes finer

---

1. Here wavelength range is calculated for 1 to 15 MHz instead of 2 to 15 MHz as most hospital ultrasound probes operate in is because this thesis describes an apparatus that uses an 1 MHz crystal.

2. The driver system used in this thesis is an 1 MHz transducer driven by one cycle per pulse, as more details will be discussed in a later section in this chapter.
as the wavelength decreases (or as the transducer frequency increase). For computer aided prosthetic socket design, 0.77 mm axial resolution is sufficient.

Although pulse repetition rate, or how frequently the driver system shocks the transducer with an electric voltage, is usually a function of transducer frequency in commercial use, this thesis does not optimize pulse repetition rate due to the scope of its objectives. It is important to adjust pulse repetition rate according to the transducer frequency to ensure that all echoed signals have been received before the next pulse is generated. Overlapping signals not only increases noise but also confuses the user of what they see. This thesis uses a trigger rate of 1000 Hz (triggering every 1 ms), which is a significantly longer period compared to the echoed sound wave\(^3\), thus a sufficiently low pulse repetition rate.

### 2.2.2 Transducer Diameter

A basic transducer is as shown in figure 2-1, that consists of matching layer, piezoelectric crystal, backing material, acoustic insulator, electrical shield, case, and wire [5]. The matching layer optimizes acoustic impedance of the transducer and the medium [19]; the piezoelectric crystal transfers mechanical and electrical energy; and the backing material provides damping to minimize ringing after pulse [19]. These physical characteristics of transducers are often selectively specified in the product datasheet.

Another important physical characteristic of the transducer is its diameter. The diameter of the transducer, together with its frequency, determines its focal depth. The sound beam of a transducer is often divided into three anatomically distinctive regions: near zone, far zone, and focal zone. The beam converges in the near zone, and is the most concentrated at the focus in the focal zone; as the wave propagates away from the focus out of the focus zone into the far zone, the beam diverges. The focal zone is a region around the focus where the reflections are more accurate than those from other depths. Half of the focal zone is in the near field, and the other half

\(^3\)The sound wave echo completely goes to zero by 0.4 ms in the setup of this thesis. The number 0.4 ms is obtained empirically.
Transducer frequency, diameter, and transducer type are crucial design considerations. Transducer frequency is determined by the crystal, which is the component left to the spring in this figure. Transducer diameter is height of the crystal in this figure; although not all crystals are circular. The structure of the transducer vary depending on the type of the transducer. Other types such as dual element transducers observe a more complex structure compared to that in this figure although they operate using the same fundamental principles.

is in the far field [5]. Because of the variations within the near field, accurate analysis can be difficult [16]. In soft tissue, the focus is given by equation 2.3\(^4\).

\[
N = \frac{D^2 \times f}{61.6}
\] (2.3)

where \(N(cm)\) is the natural focus of the transducer, \(D(mm)\) is the diameter, and \(f(MHz)\) is the frequency of the transducer. [5].

### 2.2.3 Transducer Categories

Most medical ultrasound probe has arrays of ultrasonic crystals built in [23] and has categories such as linear, convex, phased, array, etc. Ultrasonic transducers, on the other hand, have different categories such as contact transducers, dual element transducers, angle beam transducers, delay line transducers, etc. Different categories

---

\(4N = \frac{D^2 \times f}{A}\) is a more general equation where \(N[m]\) is the natural focus of the transducer, \(D[m]\) and \(f[MHz]\) are the diameter and the frequency of the transducer, and \(\lambda[m]\) is the wavelength in the medium.
offer different advantages. For instance, for near surface detection, dual element transducers has less noise due to pulse artifact while delay line transducer improves near-surface detection by adding additional materials between the crystal and the contact surface.

2.3 Driving and Receiving

Commercial products are available to drive the ultrasonic transducers and process the received signals. This thesis uses a JSR DPR300 pulser/receiver.5

In the applications of tissue boundary/tissue thickness detection, piezoelectric ultrasonic transducers are operated as following: the pulser triggers the ultrasonic transducer by electrically shock it with a pulse of high voltage. Immediately after the trigger, the pulser/receiver eliminates ringing by providing damping. The echoed ultrasonic signal becomes an electrical signal upon reception by the ultrasonic transducer. This signal is usually extremely weak, in the sub-millivolt order of magnitude. This signal is first filtered to center around the transducer frequency, then amplified for output.

The pulser determines the amplitude, pulse repetition rate, the number of pulse cycles, [5] damping, and the power in the pulse. The receiver determines the filter setting and the gain.

---

5Such ultrasonic transducer driver/receiver systems are often referred as the pulser/receivers.
Chapter 3

System Design

This chapter describes the system design from the following four aspects: overall system structure, the transducer of selection, a probe assembly and its procedure of use, and two modes of user interface. First, the system structure of sensors, data acquisition, and communication are described. Following the overall system structure, the analysis of the ultrasonic transducer of selection is discussed. Then, a hand-held probe assembly apparatus that centralizes all sensors and the microcontroller is described with design details and use procedures for tissue boundaries detection and indentation experiment. Finally, this chapter also describes the two modes of data logging, one for familiarizing the user with ultrasonic transducer measurements, and one for more efficient data acquisition.

3.1 Overall System Structure

As shown in figure 3-1, the system contains five major components: a PC, a data acquisition system (PicoScope), a pulser/receiver, a probe assembly, and a microcontroller. Table 3.1 contains the part/device numbers of each component and their description. In a probe assembly, an ultrasound transducer measures the tissue boundaries and the tissue depths; a force sensor measures the force for FEA modeling; and an accelerometer measures acceleration for self-weight compensation.

While data collection parameters such as the number of total set of data and
pre-processing thresholds, are directed instructed from the PC end, the driving and receiving parameters, such as filter frequency, gain, and damping factor, of the ultrasonic transducer is controlled by the knobs in the pulser/receiver\(^1\). The PicoScope and the microcontroller are independent data acquisition systems but data logging is centralized in the PC.

![System Diagram](image)

**Figure 3-1: System Diagram**

This system uses a PC that is connected to two independent data acquisition systems (a PicoScope and a microcontroller), which separately collects the ultrasound signal and force, acceleration signals. All sensors as well as the microcontroller are mounted to a probe assembly for ease of use.

<table>
<thead>
<tr>
<th>Parts/Device</th>
<th>Parts/Device Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pulser/Receiver</td>
<td>JSR DPR300</td>
<td>High voltage pulser and low noise receiver</td>
</tr>
<tr>
<td>Data Acquisition System</td>
<td>PicoScope 2205A</td>
<td>Digital oscilloscope with programming support</td>
</tr>
<tr>
<td>Ultrasound Transducer</td>
<td>Olympus C548-SM</td>
<td>1MHz 10mm diameter angle beam transducer</td>
</tr>
<tr>
<td>Force Sensor</td>
<td>SingleTact S8-10N</td>
<td>10N capacitive force sensor</td>
</tr>
<tr>
<td>Accelerometer</td>
<td>Adafruit ADXL326</td>
<td>5V triple axis accelerometer (+-16g)</td>
</tr>
<tr>
<td>Microcontroller</td>
<td>Arduino Nano</td>
<td>Arduino Nano 3.0</td>
</tr>
</tbody>
</table>

\(^1\)With this particular prototype, the pulser/receiver is set to have 55 dB gain, 1 to 3MHz band pass filter, PRF rate of 1, pulser amplitude of 10, pulser energy of low 1, and damping of 10 in echo mode.
### Table 3.2: Transducer Characteristics

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Diameter [mm]</td>
<td>10</td>
</tr>
<tr>
<td>Frequency [MHz]</td>
<td>1</td>
</tr>
<tr>
<td>Natural Focus in Soft Tissue [mm]</td>
<td>16.23</td>
</tr>
<tr>
<td>Axial Resolution</td>
<td>0.77</td>
</tr>
</tbody>
</table>

A 1 MHz transducer of 10 mm diameter is used in this prototype. This is a transducer suited for tissue depth of 16.23 mm or beyond. Two tissue boundaries of 0.77 mm apart or further can be distinguished by this transducer. Such transducer is a basic low profile transducer that is easy for modeling and system design.

### 3.2 Transducer Selection

In this thesis we used an 1 MHz angle beam transducer of 10mm diameter (C548-SM from Olympus) for measurement between 20 and 200mm. This section discusses this particular transducer specifications for tissue boundary and depth measurement and transducer selection parameters for reference and future work.

#### 3.2.1 Transducer in Probe Assembly

The specifications of the ultrasonic transducer used in this thesis is as in Table 3.2, calculated with the diameter and frequency specified by the vendor. Since the focus of the transducer is 16.23 mm, this transducer is intended for detection of tissue depths beyond 20 mm².

#### 3.2.2 Further Transducer Selection

For lower limb prosthetic sockets, the nearest distance between a bone and the skin is the tibia-to-skin distance, where at the thinnest regions are of sub-5mm depth. For accurate tissue boundaries detection of these regions, a transducer of focus 5 mm or closer is desirable. To achieve a 0.5 mm axial resolution, by equation 2.1 and equation 2.2, one needs a minimum 5 MHz transducer. With a 5 MHz transducer, by equation 2.3, one needs a transducer of diameter 2.5mm or less.

However, table 3.3 is merely a possible specification, and it might be difficult to

---

20mm is 16.23mm rounded up to the nearest 10s to remove ambiguity at the focus.
Table 3.3: A Possible Transducer for Near-skin Tissue Boundaries Detection

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diameter [mm]</td>
<td>2.5</td>
</tr>
<tr>
<td>Frequency [MHz]</td>
<td>5</td>
</tr>
<tr>
<td>Natural Focus in Soft Tissue [mm]</td>
<td>5</td>
</tr>
<tr>
<td>Axial Resolution</td>
<td>0.5</td>
</tr>
</tbody>
</table>

For near-skin tissue boundaries detection, a transducer of higher frequency is necessary. To meet design requirements of certain tissue depths and axial resolution, one may choose based on the commercially available sensors. A possible transducer is listed above with parameters optimized for a 5 mm or beyond tissue depths with 0.5 mm axial resolution.

find such a commercial transducer available. Depending on availability, one might trade off focus, diameter, and axial resolution.

3.3 Design of Probe Assembly

A probe assembly is designed to make it a hand-held apparatus fitting the selected components compactly into one assembly, shown in figure 3-2a and figure 3-2b. In this design, the ultrasonic transducer is aligned with the force sensor in the axial direction of the probe assembly. The force sensor is pre-loaded with a rubber pad where fixed pressure is applied by Loctite fixed screws as shown in figure 3-2c. Further mechanical design considerations are discussed in [12].

4 The probe is designed in collaboration with colleague Steven Keyes, and is modified since shown in [12] in the following ways only: the handle is extended to hold the Arduino Nano as well; a single-element configuration of the crystal is used therefore the second crystal is removed.

5 This rubber pad introduces non-linearity to the calibrated force sensor, causing non-desirable stability issues.
The probe assembly compacts all three sensors as well as the microcontroller into one handheld device for ease of use.

Figure 3-2: The Probe Assembly
3.4 Use Procedures

Different techniques are required in using the probe for tissue boundaries detection and for indentation experiments.

3.4.1 Tissue Boundaries Detection Procedure

For tissue boundaries, the following steps shall be followed as illustrated in figure 3-3:

1. Generously apply ultrasound gel at the location of choice on the limb.
2. Lightly touch the limb with the probe head, adjust by sliding and rotating, to make a direct contact between the probe head and the skin.
3. Rotate the probe around the limb in the plane of the cross-sectional view from the top to an extreme where the majority of the probe is touching the skin.
4. Start data logging on PC.
5. Slowly rotate the probe around the limb in the plane of the cross-sectional view from the top to the other extreme where the majority of the probe is touching the skin.

3.4.2 Indentation Experiment Procedure

For indentation, the following steps shall be followed as illustrated in figure 3-4:

1. Generously apply ultrasound gel at the location of choice on the limb.
2. Start data logging on PC. Be sure that the probe is not touching anything at start for calibration purposes.
3. Lightly touch the limb with the probe head, quickly adjust by sliding and rotating, to make a direct contact between the probe head and the skin.
4. Pressing into the limb at speed of experiment requirement.
5. Repeat at different indentation speed as necessary.
Figure 3-3: Using the Probe for Tissue Boundaries Detection

Tissue boundaries detection by slowly rotating the probe head around the limb. Note that the illustration is not to scale.

Figure 3-4: Using the Probe for Indentation

Indentation experiment by pressing the probe into the limb. Note that the illustration is not to scale.

3.5 Data Logging

Ultrasound, force, and acceleration data is logged at the PC end via MATLAB based PC-to-Arduino and PC-to-PicoScope interfaces. For Arduino, MATLAB Support
Package for Arduino Hardware version 17.1.1 is installed; reading requests are directly made from MATLAB. For PicoScope, the PicoScope SDK and PicoScope 2000 Series MATLAB Generic Instrument Driver version 1.8 were installed. Simple triggering for block data mode is used to acquire chunks of waveforms. The trigger is set to DC coupling, \(+/-\) 5V range, 0.1 ms time interval, and 5004 data points to collect. The sampling rate is thus 50.04 MHz, which is well beyond the Nyquist frequency 2 MHz for an 1 MHz transducer\(^6\).

Just like hospital standard ultrasound machines requires dedicated training, ultrasonic transducers require getting used to it. “Practicing” with visual feedback for using ultrasonic transducers is helpful to obtain the most meaningful signal. Further implications of the need for training will be discussed in Chapter 5.

The apparatus in this thesis has two modes: the visual feedback mode and the blind mode. In visual feedback mode, pre-processed waveforms are displayed, together with reference lines including peaks over predetermined thresholds and areas above thresholds such as in figure 3-6. This is the “training mode” where user learns how signal strength is associated with how much ultrasound gel, how much pressure, and transducer motion. After the user becomes familiar with using the transducer, blind mode is available for an improved frame rate, i.e. more data sets are recorded for the same period of recording time.

### 3.5.1 Visual Feedback Mode

As shown in figure 3-5, visual feedback mode, once data logging begins, PC requests a “block” of waveform from PicoScope, force and acceleration data from the Arduino, and a time-stamp. Before the next round of data acquisition, the waveform signal is processed and displayed.

The 0.1 ms long waveform captures a single pulse and its echo. A series of steps are taken to determine if the captured waveform is a) properly triggered, b) contains information regarding tissue boundaries/depths. If the waveform is either poorly triggered or doesn’t contain useful information, the waveform and other data at this

\(^6\)In future work, one may use a lower sampling frequency for memory saving.
In visual feedback mode, pre-processed waveforms with reference lines of peaks and above threshold regions are plotted on the PC to familiarize the user with using the apparatus.

timestamp is discarded. The waveform, together with the processed waveform is then displayed on the PC on a rolling basis as more waveforms come.

Figure 3-6 shows an example of the displayed waveform. The top subplot is a typical waveform in-vivo. The burst after 0 ms is the artifact from the triggering of the transducer. The smaller bursts of peaks between 0.03 and 0.08 ms are echoed waves at various boundaries. Depending on the angle and contact, the amplitude of the peaks vary significantly even if the overall direction maintains the same. These are likely to be echoes from fat-to-tissue, tissue-to-bone, and other boundaries where acoustic properties vary. In this particular waveform, there is one echoed wave that is significantly bigger than the other around 0.55 ms, and is in fact, indeed the tissue-to-bone boundary as verified by MRI. However, it is common to observe multiple big waves in amplitude, or none. What’s important is that the absolute value of the echoed wave amplitude does not infer what boundary it is.

The bottom subplot is the upper envelope of the filtered waveform on the top with reference lines. Note that the x-axis is the “distance” instead of time. The distance
is calculated as in equation 3.1 where $d$ is the distance between the transducer and the location where the sound echoed, $c$ is the speed of sound (1540 m/s in soft tissue), and $t$ is time. The red square are locations where the upper envelope exceeds certain threshold\(^7\), and the blue line is the peak(s) in the upper envelope that exceeds another threshold (although often set to the same value). The significance of these reference lines and further details on finding tissue boundaries using these waveforms are discussed in Chapter 4. The part of the envelope less than 20 mm is removed for processing purposes.

\[
d = c \times t/2 \quad (3.1)
\]

Due to the processing and display, the frame rate (how often a new block of waveforms is captured) is significantly slower compared to the blind mode. Since in processing, non-valid waveforms are discarded, the time to capture 500 valid data sets (waveform, force, acceleration, and timestamps) varies. On average, it takes about 36 seconds to capture 500 data sets, which rounds to approximately 14Hz.

\(^7\)The red squares in the figure are the “detection” multiplied by the threshold. The “detection” is a binary recording.
3.5.2 Blind Mode

Once the user is familiar with using the probe, blind mode can be used for an improved frame rate and thus overall more data in a given time, making experiments shorter and easier. As shown in figure 3-7, blind mode is simply visual feedback mode without preprocessing and displaying.

In blind mode, this thesis typically captures 700 samples per trial, which takes approximately 14 seconds. That is 50 Hz, which is much faster compared to 14 Hz in visual feedback mode. The frame rate is constrained to MATLAB interface, two separate communication destinations, and a long pipeline. Further details of future work to improve the frame rate is discussed in Chapter 6.

![Blind Mode Diagram](image)

**Figure 3-7:** Blind Mode Diagram
Chapter 4

Signal Processing

This chapter describes the processing of ultrasonic signal, and acceleration signal for self-weight compensation. General ultrasound waveform processing is described followed by additional rules for indentation sensing and a histogram and edge based method for tissue boundaries detection with a single-element ultrasonic transducer.

4.1 Ultrasound

Medical ultrasound typically assumes a fixed speed of sound in the human body [22]. With this assumption, one can calculate the distance between the transducer and the boundary where sound is echoed by measuring the time lapse between the trigger and the received echo. However, due to noise, multiple reflection sites, and non-direct relationship between the wave amplitude and the type of boundary it’s associated with, and a one-dimensional narrow “field of view”, it is difficult to locate the bones in the limb with “screenshots” of measurement. In particular, to detect the bone location using a single frame of waveform, it requires the probe axis to be aligned with the bone whose location is unknown to the user.

Indentation experiments are easier to process since one can simply focus on any boundary of a significant echo amplitude. One particularly reliable echo site is the tissue-to-skin boundary going out of the limb\(^1\). Although when directly facing a

\(^1\)This is, when the ultrasonic wave enters the limb, goes through the limb, and exits the limb on
bone, the described boundary will no longer be “visible” to the transducer, targeting an “open” area is often easier (for the user) than targeting a bone, although both cases are equally valid and will observe the same indentation results.

Bone location detection, on the other hand, is more difficult and requires either the probe to be aligned with the bone or an exhaustive “view” of the limb. Empirical experiences show that attempting to find the bone with a visual feedback is sometimes difficult and non-repeatable. Therefore, in an attempt to make the bone location detection procedure more user friendly, data driven, and more automatic, this thesis presents a histogram and edge based method. Further verification experiments will be discussed in Chapter 5.

4.1.1 General Waveform Processing

On the pulser/receiver side, the waveforms are bandpass filtered and amplified. In post processing on the PC, the waveforms are processed in the same way as in the visual feedback mode described in Chapter 3.

A 128th order bandpass FIR filter in the shape of a Hamming window with boundaries at 0.9 and 1.1 MHz is used to filter the waveform. Although the pulser/receiver has built in filter, it is often in a wide bandwidth. Further filtering in post processing may not be necessary but certainly removes additional noise.

The upper envelope is then abstracted from the filtered waveform to make thresholding easier and removes ambiguity in peak detection. In the filtered waveform, the exact location of peaks may vary by one or more cycles\(^2\) within the echoed wave even when the overall shape matches. The envelope returns are more consistent result in similar waveforms.

The root-mean-square(RMS) level of the upper envelope of distance beyond a distance threshold of 20mm is calculated. When the RMS level is below 1000\(^3\) (often

---

2 A cycle here refers to one going up and down in the raw ultrasound signals. At a boundary when the ultrasound wave echoes, there are typically multiple cycles in the echo.

3 The number 1000 is empirically determined and may vary with different transducers and pulser/receivers.
because either there is a bone very close to the skin or if the transducer is not in contact with the limb), the values in the upper envelope of distance between 0 to 10 mm is set to 0; otherwise the values in the upper envelope of distance between 0 to 20 mm is set to 0. This procedure is to improve near-surface detection, although a dedicated transducer is likely a better solution. When the bone is close to the skin, the sound wave quickly gets echoed and no further signal will be received. The received waveform typically observes two peaks side by side, sometimes overlapping with ambiguous peaks. The first peak is the artifact of trigger and the second is the echo from the bone. When the transducer is not in touch with anything but has a layer of ultrasound gel, similar waveforms may be observed. This is due to the echo at the ultrasound gel-to-air boundary.

4.1.2 Processing Method For Indentation Experiments

For in-vivo indentation experiments, at the beginning the furthest edge is recorded and used as “depth”. In processing a new waveform, the edge that is the closest to the previous “depth” is recorded to be the new depth. This is because in indentation experiment, movements are made significantly slower than 50 Hz, and thus by using the previous value, a much more accurate indentation is obtained.

For tissue-mimicking silicon gel indentation experiments, the echo signal is typically very clean and the wave with the greatest amplitude is returned at the hard surface boundary. Thus, in indentation experiment waveform processing, the “depth” is simply determined by the peak of the waveform of the greatest amplitude.

4.1.3 A Histogram and Edge Based Method for Tissue Boundaries Detection

As specified in chapter 3, section 3.4, in tissue-boundaries detection, we use the probe to “sweep” across the limb, through which process, a cross-sectional “view” (except for the sides) is obtained. In this process, a series of (700 as in the experiments as described in Chapter 5) waveforms are recorded. The waveforms are processed as
described in subsection 4.1.1, and two plots are generated for determination of the bone locations.

For skin-to-bone depth detection, all locations in the upper envelope whose values exceed a threshold is saved in an array, “detection”, as the red square lines in the bottom subplot in figure 3-6. In the form of a binary matrix, the “detections” of all waveforms are saved. Meanwhile, peaks in the upper envelope that exceeds a threshold (not necessarily the same as the one for “detection”) are also found and saved in the form of an array, “edges”, as the blue line in the bottom subplot in figure 3-6. As shown in figure 4-1 below, the top subplot is a plot of accumulated “detections”. The bottom subplot is a histogram of the “edges”.

![Detect / Above Threshold](image)

**Figure 4-1:** An Example Trail Processed with the Histogram and Edge Based Method

The histogram and edge method uses the accumulated detection graph as in the top subplot to determine the one or two peaks that are most likely to represent bone depth. The two most prominent peaks after the boundary peak(right most peak) are likely bone depths representations. This method then looks at the associated depth window in the edge histogram. The corresponding bone depths are the most likely depths on the left side of the corresponding window in the histogram.

The top subplot indicates where the significant echoes are as the probe “sweeps” across the cross-section of the limb. The peaks in this plot are places where are sta-
tistically likely to have substances that cause reflection of sound waves. The example in figure 4-1 is typical in lower limb tissue boundaries detection. In this particular trial, the block around distance at 100 mm is where the wave echoes at the tissue-to-skin boundary when the sound wave propagates out of the limb, the blocks at 40 to 50 mm and 70 to 80 mm are reflections at the tibia and fibula. The block at 20 to 30 mm are artifacts from echos and other sources. Typically the furthest one is always the tissue-to-skin boundary on the way out of the limb, and the next furthest one of two (depending on pre-knowledge of how many bones there are in the limb of measurement) are the tissue-to-bone boundaries.

From the upper subplot one can determine that tibia and fibular are, from the probe, somewhere between 40 to 50 mm, and 70 to 80 mm away respectively. Having the window in mind, we look at the histogram of the edges in the bottom subplot. As the probe “sweeps” across a particular bone, the distance is the shortest when it directly faces the bone. Therefore, in a block in a histogram, we want the most likely one that are the shortest, i.e. the highest on the left side of a block. In the example in figure 4-1, for the block in the window of 40 to 50 mm, we then would choose 43.13 mm, and in the block in the window of 70 to 80 mm, we would chose 70.37 mm. These are the peaks in the respective blocks who are on the left side. Statistically they represent the most likely edge within the window that are the shortest. Recall that edges as shown in figure 3-6 in blue line, are the peaks in the waveforms, and the time lapse (that is converted into distances in these plots) is fundamentally how ultrasound detects acoustic boundaries, or in our application, the distance from the location of probing on the skin to the respective bones inside of the limbs.

4.2 Self-Weight Compensation

In indentation experiments, force-depth history is measured. Here “force” is the force exerted on the skin by the probe head. However, the weight of the probe adds additional force to the force sensor as it determines the force from the probe to the skin in indentation experiments. Accelerometers are used to determine the tilt angle
of the the probe to compensate for this effect.

### 4.2.1 Calibration

The force sensor and the accelerometer both require calibration due to difference among sensors out of production.

Although the SingleTact force sensor comes with a calibration PCB and exhibits a linear behavior to force, in this assembly, the rubber pad in pre-load causes additional non-linearity and thus requires the force sensor to be calibrated after assembly of the probe. This force sensor is calibrated with calibration weight, and force is determined by a look-up table generated in calibration\(^4\).

Gravity is used as reference for the calibration of the accelerometer. The recorded raw data are as in Table 4.1 for the specific accelerometer in this assembly.

<table>
<thead>
<tr>
<th></th>
<th>Maximum Reading [V] at +g</th>
<th>Minimum Reading [V] at −g</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>2.1457</td>
<td>1.3539</td>
</tr>
<tr>
<td>y</td>
<td>2.1114</td>
<td>1.3148</td>
</tr>
<tr>
<td>z</td>
<td>2.2581</td>
<td>1.3832</td>
</tr>
</tbody>
</table>

**Table 4.1: Accelerometer Raw**

### 4.2.2 Self-Weight Compensation

With a 3-axis accelerometer, tilt can be determined with improved sensitivity and accuracy combining x, y, and z signals. Pitch, roll, and tilt are calculated in the following equations [24].

\[
Pitch = \alpha = \arctan\left(\frac{x}{\sqrt{y^2 + z^2}}\right)
\]  

\[
Roll = \beta = \arctan\left(\frac{y}{\sqrt{x^2 + z^2}}\right)
\]

\(^4\)Although this particular force sensor in this particular probe observes a significantly lower repeatability rate than design requirement as in [21], it can be easily improved in next generation prototypes by using a load cell instead, such as shown in [7].
\[ \text{tilt} = \gamma = \arccos\left(\frac{\hat{z}}{g}\right) \quad (4.3) \]

where \(x, y, z\) are the accelerometer readings as labeled, \(g\) is the gravity constant, and \(\alpha, \beta, \gamma\) as shown in figure 4-2.

![Figure 4-2: Tilt angles from a tri-axis accelerometer as in [24]](image)

The title angle (the angle between handle of the probe and the direction perpendicular to contact surface) is \(\beta\) as in 4.2, as shown in figure 4-3. The additional force by the probe weight on the force sensor to subtract is then given by

\[ w_c = w_2 \times \cos \beta \quad (4.4) \]

if \(0 < w_c < \frac{\pi}{2}\), and

\[ w_c = w_1 \times \cos \beta \quad (4.5) \]

if \(\frac{\pi}{2} < w_c < \pi\), where \(w_c\) is the weight to compensate, \(w_1\) is the head weight (probe head till force sensor), \(w_2\) is the tail weight (force sensor till tail of the probe).
Figure 4-3: Accelerometer Directions as in the Probe
Chapter 5

Experiments and Verification

To verify that the system design meets the objectives of an accurate tissue boundary and indentation detection using low profile ultrasonic transducers, \textit{in-vivo} experiments are performed and results are compared to "ground truth" from MRI scans and DIC results, respectively. To understand the limits of ultrasonic sensing for this application, the tissue boundary detection results are also compared to that from a commercial ultrasound imaging system. Furthermore, a phantom staircase study for both depth and indentation are performed to remove the inaccuracy introduced by human factors in \textit{in-vivo} experiments.

In this chapter, two sets of four experiments are described: depths and indentation experiments in a phantom staircase, and bone depths and indentation experiments \textit{in-vivo}. Following the experiment methods and results, this chapter discusses sources of errors in the system, for phantom gel and for human tissue.

5.1 Depth Sensing Verification by a Phantom Staircase

To test the accuracy of depth measurement as a result of a single ultrasonic transducer, a 19.7cm x 9.4cm x 5.6cm "staircase" filled silicon gel (the phantom) that exhibits a similar mechanical property\cite{15} as soft tissue is built.
As shown in figure 5-1, a staircase of known depth is filled with phantom gel (SYLGARD 527 A&B Dow Corning, MI, USA, A:B ratio 1:1, cured in incubator set to 60°C for four hours). While the walls of the staircase is Acrylic, the steps are 3D printed with VeroClear (Stratasys). The steps are designed to have 10 mm increment; and the staircase is filled 4 mm (manually measured by a ruler) to the top such that the distances from the phantom surface to the bottom of the steps are 6mm, 16mm, 26mm, 36mm, 46mm, 56mm, 66mm, 76mm, and 86mm. This distance is manually verified by a ruler from the outside of the transparent Acrylic wall.

![Phantom in a Staircase](image)

*Figure 5-1: Phantom in a Staircase*

This phantom staircase is used for both depth and indentation verification.

A group of preliminary data is collected to determine the speed of sound in the phantom. In each step from 16 to 86 mm, the time lapse is manually measured in an digital oscilloscope\(^1\). The speed of sound is calculated by linearly fitting to the “truth depths” using a least squares method (see Appendix). The results of the measurements and fitting is plotted in figure 5-2. The speed of sound is determined to be 1007 m/s.

A group of data is then gathered from the step of 16mm distance to the step of 86mm to determine the accuracy of depth detection and repeatability. In each step, 4

\(^1\)Manual measurements were made with the foreknowledge of the fixed step incremental distance.
Preliminary data is acquired from steps of depths 16mm to 86mm. The time lapse is recorded for each step. The results are linearly fitted to estimate the speed of sound in this particular phantom to 1007 m/s.

Trails are run, and results are plotted in figure 5-3a, the error distribution is plotted in figure 5-3b and detailed in table 5.1. As shown in figure 5-3a, the measurements appear accurate with an overall 0.36 mm mean error and 1.1 mm standard deviation. The mean and standard deviation appear depth independent although standard deviation observes an relative increase in the final step. This might be due to the interference from the echo at the corners.

Table 5.1: Phantom Staircase Depth Detection Experiment Result

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean error</td>
<td>0.3594 [mm]</td>
</tr>
<tr>
<td>Max absolute error</td>
<td>2.3000 [mm]</td>
</tr>
<tr>
<td>Min absolute error</td>
<td>0 [mm]</td>
</tr>
<tr>
<td>Standard error deviation</td>
<td>1.1086 [mm]</td>
</tr>
</tbody>
</table>
(a) Phantom Staircase Depth Detection Results

The error bar plot of the measurements in each step from depth 16mm to 86mm; four trials per step.

(b) Phantom Staircase Depth Detection Error Histogram

**Figure 5-3:** Phantom Staircase Depth Detection
5.2 Indentation Verification by DIC in Phantom

To verify depth detection in indentation experiments, experiments are conducted in the phantom staircase. A group of four experiments each with four trials are conducted at steps of depth 26, 46, and 76 mm with a repeated trial at depth 46 mm. These are representative of shallow, medium, and deep steps. Two Go-Pro cameras are used to capture pairs of pictures in sync every five seconds\(^2\). The probe assembly and the limb were labeled with black dots of 5 mm diameter for image processing purpose as shown in figures in figure 5-4. The indentation experiment uses procedure as described in chapter 3 section 3.4.2. A stopwatch is used to sync the DIC results with my system results\(^3\).

![Camera Views of DIC Experiments in Phantom](image)

(a) Left Camera View of Phantom Indentation Experiment  (b) Right Camera View of Phantom Indentation Experiment

Figure 5-4: Camera Views of DIC Experiments in Phantom

The DIC algorithms\(^4\) takes user selected dot groups from left and right cameras, and compute the 3D position of the centroid of each dot in each frame. The position

---

\(^2\)Due to the availability of DIC setup, five seconds apart is the shortest window.

\(^3\)As the system starts to record, PC interface displays “Recording started”. Upon seeing the message, the stopwatch is manually started and held in view of the cameras.

\(^4\)DIC processing is in collaboration with colleague Dana Solav.
of the probe head is then calculated based on the dots' positions and the distance from the dots to the probe head. Finally, the indentation is calculated by finding the projection of translation of the probe head position with respect to the direction of the probe handle in the first set of images as in equation 5.1.

\[ d = \overrightarrow{\tau} \cdot \overrightarrow{AB} \]  

(5.1)

where \( d \) is indentation in this frame, \( \overrightarrow{\tau} \) is the normalized vector pointing in the direction of the probe handle, \( \overrightarrow{A} \) is the position of the probe head in this frame, and \( \overrightarrow{B} \) is the position of the probe head in frame No.1\(^5\).

The results obtained from my system are compared to that from DIC as shown in figure 5-5 and table 5.2. In these trails, the results observes a similar mean (-0.1289 vs. 0.3594 mm) and standard deviation (1.3826 vs. 1.1086 mm) of error as compared to depth sensing results in the phantom.

![Error Histogram of Indentation Experiment in Phantom using DIC result as ground truth](image)

**Figure 5-5:** Phantom Staircase Indentation Experiment Error Distribution

<table>
<thead>
<tr>
<th>Table 5.2: Phantom Staircase Indentation Experiment Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean error</td>
</tr>
<tr>
<td>Max absolute error</td>
</tr>
<tr>
<td>Min absolute error</td>
</tr>
<tr>
<td>Standard error deviation</td>
</tr>
</tbody>
</table>

\(^5\)Although this method may not be the most accurate method to process the images but given that the probe head stays relatively still (indentation is always below 10mm) and does not typically observe too much rotation, this method should return a reasonable result.
Figure 5-6 shows an example of an indentation experiment. As indentation increases, the depth is decreasing while the force\(^6\) at the probe tip was increasing. The title angle, calculated from acceleration data, shows that through this process, the probe was held relatively still.

![Figure 5-6: Example Data of Depth, Force, and Acceleration Reading in an Indentation Experiment](image)

**5.3 Indentation Verification by DIC *in-Vivo***

With a narrow error band shown in phantom indentation experiments as described above, a group of four trials are run *in-vivo*. At the same location in a lower limb in a healthy individual, four trials of indentation experiments were performed with the same DIC setup as described in phantom indentation verification experiments. The views of the cameras as shown in figure 5-7.

\(^6\)As described in previous chapters, the force reading is inaccurate due to probe design. This is a demonstration that force sensing and indentation sensing can be formed using this apparatus possibly with an improved force sensor mounting.
The system results are compared to the DIC results, shown in figure 5-8 and table 5.3. Compared to the phantom indentation experiments, in-vivo results show a slightly greater mean error (by 0.06mm), and a narrower standard deviation (by 0.9mm). This can be interpreted as the two results show similar accuracy both in-vivo and in phantom.
5.4 A Comparison with a Commercial Ultrasound System with Respect to “Ground Truth” by MRI in-Vivo

To test the accuracy of skin-to-bone distance measurement in-vivo, we compared the accuracy of results obtained by method as described in Chapter 4 in five distinctive locations on the lower limb of a healthy individual, and compared results to that from a commercial ultrasound imaging instrument using MRI scan measurements as “ground truth”.

Five MRI markers around the center of the limb distributed around the limb are attached to the limb as the MRI scan is taken, as shown in figure 5-9a. The location of the MRI markers are then marked with temporary tattoo ink to be consistent in the following experimentations.

The distances from each marker location to tibia and fibula are measured from the MRI scan by manually selecting the line of measurement and converting to distance in the physical world shown in figure 5-9 (b) to (f). The measurement data are shown in table 5.4. In the following experiments, the distances obtained from the MRI scan as shown in table 5.4 are considered the “ground truth”, or “true depths”.

<table>
<thead>
<tr>
<th>Marker Number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest Distance to Tibia from Marker [mm]</td>
<td>12</td>
<td>28.8</td>
<td>64.2</td>
<td>73.4</td>
<td>46</td>
</tr>
<tr>
<td>Longest Distance to Tibia from Marker [mm]</td>
<td>28</td>
<td>63</td>
<td>75.5</td>
<td>80.77</td>
<td>56.5</td>
</tr>
<tr>
<td>Shortest Distance to Fibula from Marker [mm]</td>
<td>43</td>
<td>63</td>
<td>75.8</td>
<td>41.23</td>
<td>24.1</td>
</tr>
<tr>
<td>Longest Distance to Fibula from Marker [mm]</td>
<td>51.3</td>
<td>66</td>
<td>84.7</td>
<td>52</td>
<td>30.1</td>
</tr>
</tbody>
</table>

The system measures the shortest distances from the location of probing to the bones, i.e. tibia and fibula. The longest distance to each bone are measured for the purpose of interpreting the data in the cases such as when the distance is too short for the transducer of selection or when the results are not obvious.
Figure 5-9: Paths to Tibia and Fibula from Markers

Five MRI markers are as labeled in (a). From each marker location, the distance from the marker to tibia and fibula are measured by manually clicking on the MRI image and converting the distance on the image to distance in the limb. While most are of distance beyond 20mm, some (for instance in (b)) are shorter. Those who are within the detection range are considered for data processing.
Following the procedure for tissue boundary detection as described in Chapter 3, in blind mode, four trials are run for each of the five marker locations. The shortest distances to tibia and fibula were recorded and is shown in the upper subplot in figure 5-10.

To determine how the performance compares to commercial ultrasound machines, measurements are taken using a commercial ultrasound system (Telemed SmartUs EXT-1M) in a similar manner: at each of the marker location, the probe slowly "sweep" around the limb and a movie was recorded. Using the provided software, tissue boundaries are first manually visually determined and the exact distances are then measured within the software. The results were plotted in the bottom subplot in figure 5-10.

The mean and standard deviation of overall error with respect to MRI measurement results are listed in table 5.5, and error histograms are shown in figure 5-11. The two results are very comparable and both observes mean error less than 1 mm. In this particular set of trials, our results outperforms the commercial ultrasound machine in both mean error and standard deviation of error by a narrow margin. One possible reason is that the author of this thesis has not gone through official training for ultrasound imaging techniques.

<table>
<thead>
<tr>
<th></th>
<th>My Result</th>
<th>Commercial Ultrasound Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Error mm</td>
<td>0.2256</td>
<td>0.4768</td>
</tr>
<tr>
<td>Max Absolute Error mm</td>
<td>4.8400</td>
<td>5.8000</td>
</tr>
<tr>
<td>Min Absolute Error mm</td>
<td>0</td>
<td>0.4000</td>
</tr>
<tr>
<td>Standard Deviation mm</td>
<td>2.2183</td>
<td>3.2842</td>
</tr>
</tbody>
</table>

Small error margin compared to ground truth (MRI) of sub millimeter mean error and 2 to 3 mm standard deviation. Similar results between my system and the commercial ultrasound imaging system.

Table 5.5: Mean and Standard Deviation of Error Compared to MRI Results

---

Due to unavailability of 1 MHz option in the commercial ultrasound machine available, we used 5 MHz option. 5 MHz probe, compared to 1 MHz transducer, observes a better axial resolution.
Figure 5-10: Measurement vs. MRI Results and Commercial Ultrasound Results

From the same five location as shown in the MRI scans, measurements are taken from both my system and a commercial ultrasound imaging system. The measured results are plotted against the MRI measured results with error bars.
Both error histograms have error distributions within -6 to 6mm. The distribution of error with respect to MRI results appear more close to center/zero error in my system as compared to that of the commercial ultrasound system.
5.5 Error Analysis

Although overall the observed errors are small, there are a few factors that consistently contribute to error.

5.5.1 Error in the Phantom

Compared to *in-vivo*, signals in the phantom tends to be cleaner, due to the homogeneity of the medium. However, a natural tremor in hand can easily affect the probing location, resulting in a displacement and thus different compared to the depth of the gel.

5.5.2 Error *in-Vivo*

While due to the complexity of tissue, compared to phantom gel, more echo sites and noise source is typical to tissue, there are a few error sources by user operation and system design.

**Speed of Sound Estimation**

In both commercial ultrasound machine and in the apparatus described in this thesis, a fixed speed of sound (1540 m/s in particular as in soft tissues) is assumed. This introduces error because typically for all locations in the limb there is a layer of fat. The thickness of the fat layer vary from person to person but in case that its thickness is significant, the assumption of 1540 m/s speed of sound will result in error. The speed of sound in fat is 1450 m/s [13]. In an example of the MRI as in figure 5-9a, the thickest fat layer is as thick as 13 mm. In case that the bone is 70 mm away, the time of travel is approximately 0.1 ms. The time in fat is approximately 0.02 ms. With 1540 m/s, the distance in this specific region calculates to approximately 15.4 mm while for 1450 m/s, the same time lapse calculates to 14.5 mm. Although in this rough estimated analysis the error is sub-millimeter, one can see that the

\[\text{Note that in these calculations, sound waves travel to the bone and comes back to the transducer so dividing by a factor of a 2 is necessary in calculating the depth (vs. distance the sound travels).}\]
assumption of speed of sound is surely a source of error. Similarly for someone who is very muscular, this assumption will introduce error since the speed of sound in muscle is 1550-1630 m/s[13], which is greater compared to 1540 m/s.

Training

Similar to that hospital ultrasound machines require training, ultrasonic transducer use in-vivo requires getting used to as well. The human operation introduces variables in taking the same measurement. Future work is necessary to realize a more repeatable and autonomous data acquisition.
Chapter 6

Discussion

To make computer aided prosthetic socket design more accessible, this thesis described an apparatus that can a) detects the skin-to-bone depth and b) perform indentation experiments for tissue mechanical property. Low profile sensing including single element ultrasonic sensing, force sensing, and acceleration sensing are included. This apparatus is light-weight, compact, and affordable while demonstrating results of the same standard as a commercial ultrasound imaging system in the case studies. A cost breakdown is detailed in this chapter.

![Figure 6-1: A Hand-Held Apparatus](image)

The system includes a probe assembly (as in the picture), data acquisition systems, sensor driver/receiver system, and a PC.

As shown in experimental verifications, this thesis demonstrates that ultrasonic technology is appropriate for tissue boundary detection and indentation experiment. Further more, a single element ultrasonic transducer is demonstrated enough to pro-
duce results with mean error less than 1 mm. A portable, low cost device for the purpose of tissue boundary detection and indentation for prosthetic socket design parameter measurement is proven possible.

![Figure 6-2: Estimated PDF from Four Experiments](image)

All trail results show mean error well below 0.5 mm and a standard deviation of error well below 2.5 mm. Overall the error is evenly distributed.

However, moving forward with next generation prototypes, a few design considerations are discussed in this chapter to improve accuracy, efficiency, and portability.
In this thesis, four experiments are conducted: two for depth sensing, in phantom and in-vivo, and two for indentation sensing, in phantom and in-vivo. Results show a similar distribution of error that centers around zero and has a standard deviation of less than 2 mm.
6.1 Cost Breakdown

With the effort to further miniaturize the system, in the long run, the parts of the system should roughly include an ultrasonic transducer, a reliable force sensor, an accelerometer, a PCB with a microcontroller, a battery, and its assembly. Table 6.1 below shows the breakdown of the estimated cost of each part. Overall, the system will cost roughly $1200 as opposed to about $20,000 to $75,000 for an ultrasound imaging system, and $400,000+ for an MRI scanner; not to mention its additional support for indentation experiments.

With the system described in this thesis, instead of a PCB and a batter, we have a PicoScope($139.00), a Pulser/Receiver(approx. $3000), and an Arduino Nano(approx.$4); instead of a more reliable force sensor such as a load cell, we have a force sensing capacitor(approx. $74). The total of the system as described in this thesis is approximately $3707, which is still significantly below the cost for ultrasound imaging systems and MRI machines.

Table 6.1: Cost Breakdown of the System after Further Miniaturization

<table>
<thead>
<tr>
<th>Part</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ultrasonic Transducer</td>
<td>approx. $400</td>
</tr>
<tr>
<td>Force Sensor</td>
<td>approx. $400</td>
</tr>
<tr>
<td>Accelerometer</td>
<td>approx. $20</td>
</tr>
<tr>
<td>PCB with Microcontroller</td>
<td>approx. $250</td>
</tr>
<tr>
<td>Battery</td>
<td>approx. $60</td>
</tr>
<tr>
<td>Housing/Assembly</td>
<td>approx. $70</td>
</tr>
<tr>
<td>Total</td>
<td>approx. $1200</td>
</tr>
</tbody>
</table>

A further miniaturized system will cost roughly $1200 while the system as described in this thesis costs roughly $3700.

6.2 Future Design Recommendations

The objective of this thesis is a proof of concept that tissue boundary and indentation measurement are feasible using a single ultrasonic transducer with force sensing and self-weight correction using a histogram and edge based method. Listed below are a few design considerations moving towards the next generation prototype.
1. Optimize pulse repetition rate: currently the pulse repetition rate is simply set to maximum to avoid overlapping echo. As communication bandwidth improves, pulse repetition rate shall be optimized for a better temporal resolution.

2. Another probe for near-skin bone detection: currently the transducer is optimal for detections further than 20 mm. As discussed in Chapter 2, one could either add another transducer to the assembly or use separate probe assemblies for an improved near-skin detection.

3. Increase frame rate by centralizing all components: currently the frame rate is limited by communication bandwidth, which is limited due to multiple devices in the system. One way to improve frame rate is to centralize all component such that communication or on-board data storage is performed by a single micro-controller with minimum layers in the pipeline.

4. Miniaturizing the system via hardware development: the system can become more compact and portable if a single PCB can serve the roles of pusler/receiver, data acquisition system, on-board data logging system, and communication system to PC, force sensor, and accelerometer. Preliminary work is shown in [12].

5. Minimizing speed of sound assumption error: as discussed in Chapter 5, accuracy may be improved by using a more accurate speed of sound. By looking at patient information such as BMI in combination with inference from echo close to the transducer, one may be able to improve accuracy by a better estimation of the assembly of tissue types.

6. Use load cells for force sensing.
Appendix A

Code

Unless otherwise specified at the beginning of a file, the author of all code is the author of this thesis.

A.1 Data Logging

A.1.1 Blind Mode Depth Sensing

%% Blind Mode - Depth Measurement
%% Modified based on PicoScope 2000 Series Instrument Driver Oscilloscope
   Block Data Capture with Simple Trigger Example
%% *Copyright:* 2013 - 2017 Pico Technology Ltd. All rights reserved.

%% Setup PicoScope:

%% The PicoScope 2205A uses the API functions from the ps2000 shared
   library and so requires the PicoScope 2000 Series MATLAB Generic
   Instrument Driver. Please install this using the Add-Ons Explorer
   menu item in the toolbar of the MATLAB Home tab.
%% Please note that there is an error in the PS2000Config file which
   affects path ordering - please ensure that you add the win64 folder
   from the Instrument Driver root folder to the top of the MATLAB path
This will be fixed in due course.

Also, please copy the dlls from this folder(https://drive.google.com/drive/folders/0Bz9oJypl8PmZ2R2N2WThWYXVnVzQ?usp=sharing) or (https://drive.google.com/open?id=0Bz9oJyp18PmZ2R2N2WThWYXVnVzQ) to the following location:

C:\Program Files\Pico Technology\SDK\lib

Setup Arduino: MATLAB Support Package for Arduino Hardware version 17.1.1

Declaration: this version may not be the exact code version used in gathering the data shown in Chapter 5. In particular, the threshold may vary based on setup etc.

License: MIT License, No commercial use

Clear Command Window and Close any Figures

clear

close all

clear

Load Configuration Information

PS2000Config;

Device Connection

Create a device object.

ps2000DeviceObj = icdevice('picotech.ps2000.generic.mdd');

Connect device object to hardware.

connect(ps2000DeviceObj);

Obtain Device Groups

Obtain references to device groups to access their respective properties

72
Block specific properties and functions are located in the Instrument Driver's Block group.

```matlab
blockGroupObj = get(ps2000DeviceObj, 'Block');
blockGroupObj = blockGroupObj(1);
```

Trigger specific properties and functions are located in the Instrument Driver's Trigger group.

```matlab
triggerGroupObj = get(ps2000DeviceObj, 'Trigger');
triggerGroupObj = triggerGroupObj(1);
```

Configure the Device

% Enable Channels A and B, set the sampling interval, number of samples to collect and set a simple trigger.

% Set channels:

```matlab
% Channel : 0 (ps2000Enuminfo.enPS2000Channel.PS2000.CHANNEL.A)
% Enabled : 1 (PicoConstants.TRUE)
% DC : 1 (DC Coupling)
% Range : 6 (ps2000Enuminfo.enPS2000Range.PS2000.1V)
[status.setChA] = invoke(ps2000DeviceObj, 'ps2000SetChannel', 0, 1, 1, 6);
% Channel : 1 (ps2000Enuminfo.enPS2000Channel.PS2000.CHANNEL.B)
% Enabled : 1 (PicoConstants.TRUE)
% DC : 1 (DC Coupling)
% Range : 7 (ps2000Enuminfo.enPS2000Range.PS2000.2V)
```
% [status.setChB] = invoke(ps2000DeviceObj, 'ps2000SetChannel', 1, 1, 1, 7);
% Range : 8 (ps2000Enuminfo.enPS2000Range.PS2000.5V)
[status.setChB] = invoke(ps2000DeviceObj, 'ps2000SetChannel', 1, 1, 1, 8);

%%
% Set sampling interval and number of samples to collect:

% Set sampling interval - the actual sampling interval used by the driver
% will be returned.

% timeIntervalUs : 100 (microseconds) --> 10us / 0.01 ms --> 100us / 0.1 ms
[samplingIntervalUs, maxBlockSamples] = invoke(blockGroupObj, 'setBlockIntervalUs', 0.1);

% Confirm the timebase index selected.
timebaseIndex = get(blockGroupObj, 'timebase');

% Set the number of samples to collect. 2048 -->
set(ps2000DeviceObj, 'numberOfSamples', 5004);
%%
% Set simple trigger:

% Set the trigger delay to 50% (samples will be shown either side of the trigger point, ----> Will show 10% left
set(triggerGroupObj, 'delay', -10.0);

% Set the autoTriggerMs property in order to automatically trigger the oscilloscope after 1 second if a trigger event has not occurred. Set to 0
% to wait indefinitely for a trigger event. ----> 1000 for 1 second
set(triggerGroupObj, 'autoTriggerMs', 0);
% Parameters taken from config file loaded into workspace.

[simpleTriggerStatus] = invoke(triggerGroupObj, 'setSimpleTrigger', ...
    simpleTrigger.threshold, ...
    ps2000ConfigInfo.simpleTrigger.direction);

%% get one set of data
[bufferTimes, bufferChA, bufferChB, numDataValues, timeIndisposedMs] = 
    invoke(blockGroupObj, 'getBlockData');
stopStatus = invoke(ps2000DeviceObj, 'ps2000Stop'); % Stop the Device
bufferTimes = double(bufferTimes);

timesUnits = timeunits(get(blockGroupObj, 'timeUnits')); % Find the time
    units used by the driver.
timeLabel = strcat('Time (', timesUnits, ')');

waveforms = zeros(length(bufferTimes), trialnum);

%% Data Collection
trialnum = 700;
time = zeros(size(depths));
tic; % start timer

i=1;
disp('starting data collection...');
for i = 1:trialnum

    % Capture a block of data on Channels A and B together with times.
    % Data for channels is returned in millivolts.
    [bufferTimes, waveforms(:,i), numDataValues, timeIndisposedMs] = 
        invoke(blockGroupObj, 'getBlockData');

    % Stop the Device
    stopStatus = invoke(ps2000DeviceObj, 'ps2000Stop');
time(i) = toc;
end

%% Disconnect
% Disconnect device object from hardware.
disconnect(ps2000DeviceObj);
delete(ps2000DeviceObj);

%% Save time, depths, force, and acceleration data to CSV
clc
curr-time = datetime('now');
curr-time.str = datestr(curr.time);
curr-time.str = strrep(curr-time.str, ' ', '-');
curr-time.str = strrep(curr-time.str, ':', '-');
curr-time.str = strrep(curr-time.str, '-', '-');
timeStamps = strcat(curr-time.str, '-timestamps.csv');
waveform_filename = strcat(curr-time.str, '-waveforms.csv');

pathName='C:\Users\simulator II\Documents\MATLAB\Staircase\16';
fileName=fullfile(pathName,timeStamps);
disp(fileName)

fileName.waveform=fullfile(pathName,waveform_filename);
disp(fileName.waveform)

csvwrite(fileName, time);

[r, c] = size(waveforms);
N = zeros(r, c+1);
N(:,1) = bufferTimes;
N(:,2:end) = waveforms;
csvwrite(fileName.waveform,N);
A.1.2 Blind Mode Indentation Sensing

%% Blind Mode - Indentation Experiment
%% Modified based on PicoScope 2000 Series Instrument Driver Oscilloscope
Block Data Capture with Simple Trigger Example
%% *Copyright:* 2013 - 2017 Pico Technology Ltd. All rights reserved.

% Setup PicoScope:

% The PicoScope 2205A uses the API functions from the ps2000 shared
library and so requires the PicoScope 2000 Series MATLAB Generic
Instrument Driver. Please install this using the Add-Ons Explorer
menu item in the toolbar of the MATLAB Home tab.
% Please note that there is an error in the PS2000Config file which
affects path ordering - please ensure that you add the win64 folder
from the Instrument Driver root folder to the top of the MATLAB path.
% This will be fixed in due course.
% Also, please copy the dlls from this folder (https://drive.google.com/
drive/folders/0Bz9oJypl8PmZR2N2WThWYXVnVzQ?usp=sharing) or (https://
drive.google.com/open?id=0Bz9oJyp18PmZR2N2WThWYXVnVzQ) to the
following location:
% C:\Program Files\Pico Technology\SDK\lib

% Setup Arduino: MATLAB Support Package for Arduino Hardware version
17.1.1

% Declaration: this version may not be the exact code version used in
gathering the
% data shown in Chapter 5. In particular, the threshold may vary based
% on setup etc.

% License: MIT License, No commercial use

%% Clear Command Window and Close any Figures
clc
close all
clear all
%% Load Configuration Information
PS2000Config;

%% Device Connection

% Create a device object.
ps2000DeviceObj = icdevice('picotech.ps2000.generic.mdd');

% Connect device object to hardware.
connect(ps2000DeviceObj);

%% Arduino Connection
a = arduino('com10', 'nano', 'Libraries', 'I2C');
tmp102 = i2cdev(a, '0x06');

%% Obtain Device Groups
% Obtain references to device groups to access their respective properties and functions.
% Block specific properties and functions are located in the Instrument Driver's Block group.
blockGroupObj = get(ps2000DeviceObj, 'Block');
blockGroupObj = blockGroupObj(1);

% Trigger specific properties and functions are located in the Instrument Driver's Trigger group.
triggerGroupObj = get(ps2000DeviceObj, 'Trigger');
triggerGroupObj = triggerGroupObj(1);

%% Configure the Device
% Enable Channels A and B, set the sampling interval, number of samples to collect and set a simple trigger.

%%
% Set channels:

% Channel : 0 (ps2000Enuminfo.enPS2000Channel.PS2000.CHANNEL_A)
% Enabled : 1 (PicoConstants.TRUE)
% DC : 1 (DC Coupling)
% Range : 6 (ps2000Enuminfo.enPS2000Range.PS2000.1V)
% status.setChA = invoke(ps2000DeviceObj, 'ps2000SetChannel', 0, 1, 1, 6);%
% Range : 8 (ps2000Enuminfo.enPS2000Range.PS2000.5V)
% status.setChA = invoke(ps2000DeviceObj, 'ps2000SetChannel', 0, 1, 1, 8);

% Channel : 1 (ps2000Enuminfo.enPS2000Channel.PS2000.CHANNEL_B)
% Enabled : 1 (PicoConstants.TRUE)
% DC : 1 (DC Coupling)
% Range : 7 (ps2000Enuminfo.enPS2000Range.PS2000.2V)
% status.setChB = invoke(ps2000DeviceObj, 'ps2000SetChannel', 1, 1, 1, 7);
% Range : 8 (ps2000Enuminfo.enPS2000Range.PS2000.5V)
% status.setChB = invoke(ps2000DeviceObj, 'ps2000SetChannel', 1, 1, 1, 8);

%%
% Set sampling interval and number of samples to collect:

% Set sampling interval - the actual sampling interval used by the driver will be returned.

% timeIntervalUs : 100 (microseconds) -> 10us / 0.01 ms -> 100us / 0.1 ms
[samplingIntervalUs, maxBlockSamples] = invoke(blockGroupObj, 'setBlockIntervalUs', 0.1);

% Confirm the timebase index selected.
timebaseIndex = get(blockGroupObj, 'timebase');

% Set the number of samples to collect. 2048 -->
set(ps2000DeviceObj, 'numberOfSamples', 5004);

% Set simple trigger:

% Set the trigger delay to 50% (samples will be shown either side of the trigger point. ----> Will show 10% left
set(triggerGroupObj, 'delay', -10.0);

% Set the autoTriggerMs property in order to automatically trigger the oscilloscope after 1 second if a trigger event has not occurred. Set to 0
% to wait indefinitely for a trigger event. ----> 1000 for 1 second
set(triggerGroupObj, 'autoTriggerMs', 0);

% Parameters taken from config file loaded into workspace.

[simpleTriggerStatus] = invoke(triggerGroupObj, 'setSimpleTrigger', ... 
    simpleTrigger.threshold, ... 
    ps2000ConfigInfo.simpleTrigger.direction);

% get one reading from sensor
data = read(tmp102, 6);
frameIndex = typecast([data(2) data(1)],'uint16');
timeStamp = typecast([data(4) data(3)],'uint16'); %0.1ms increments
output = typecast([data(6) data(5)],'uint16');
clc
disp(dec2hex(data));
disp(strcat('frame index: ', num2str(frameIndex)));
disp(strcat('time stamp: ', num2str(timeStamp)));
disp(strcat('output: ', num2str(output)));

%% Update baseline
% This block doesn't seem to do anything
baselineMSB = data(5);
baselineLSB = data(6);
% baseline = typecast([data(6) data(5)],'uint16');

updateBaseline = [uint8(2), ... % '0x02'
    uint8(41), ... % write offset in register block
    uint8(2), ... % number of bytes to write
    baselineMSB, ... % data to write
    baselineLSB, ... % data to write
    uint8(hex2dec('FF')) %0xFF - end of packet delimiter
    ];

write(tmp102, updateBaseline); % calibrate to baseline

pause(1);

%% Get one set of data
[bufferTimes, bufferChA, bufferChB, numDataValues, timeIndisposedMs] =
    invoke(blockGroupObj, 'getBlockData');
stopStatus = invoke(ps2000DeviceObj, 'ps2000Stop'); % Stop the Device
bufferTimes = double(bufferTimes);

timesUnits = timeunits(get(blockGroupObj, 'timeUnits')); % Find the time
    units used by the driver.
timeLabel = strcat('Time (', timesUnits, ',

waveforms = zeros(length(bufferTimes), trialnum);

%% Data Collection

81
data = read(tmpl02, 6);
baseline = typecast([data(6) data(5)],'uint16');

trialnum = 700;
force = zeros(size(depths));
x = zeros(size(depths));
y = zeros(size(depths));
z = zeros(size(depths));

force = zeros(size(depths));
tic; %start timer

% Capture a block of data on Channels A and B together with times.
% Data for channels is returned in millivolts.
[buffers, waveforms(:,i), numDataValues, timeIndisposedMs] =
invoke(blockGroupObj, 'getBlockData');

% Stop the Device
stopStatus = invoke(ps2000DeviceObj, 'ps2000Stop');

% Record force
force = read(tmpl02, 6);
force now = typecast([data(6) data(5)],'uint16');
force now = force now - double(baseline);
force(i) = force now;

% Read accelerometer
x(i) = readVoltage(a,'A0');
y(i) = readVoltage(a,'A1');
z(i) = readVoltage(a,'A2');

time(i) = toc;
%% Disconnect
% Disconnect device object from hardware.
disconnect(ps2000DeviceObj);
delete(ps2000DeviceObj);

%% Save time, depths, force, and acceleration data to CSV
clc
curr_time = datetime('now');
curr_time_str = datestr(curr_time);
curr_time_str = strrep(curr_time_str, ' ', '_');
curr_time_str = strrep(curr_time_str, ':', '_');
curr_time_str = strrep(curr_time_str, '-', '_');
timeDepthsForceAccel = strcat(curr_time_str, '_sensors.csv');
waveform_filename = strcat(curr_time_str, '_waveforms.csv');

pathName='C:\Users\simulator II\Documents\MATLAB\Staircase\16';
fileName=fullfile(pathName,timeDepthsForceAccel);
disp(fileName)

fileName_waveform=fullfile(pathName,waveform_filename);
disp(fileName_waveform)

M = zeros(length(time), 5); % time, depths, force, x, y, z
M(:,1) = time;
M(:,2) = force;
M(:,3) = x;
M(:,4) = y;
M(:,5) = z;
csvwrite(fileName, M);

[r, c] = size(waveforms);
N = zeros(r, c+1);
N(:,1) = bufferTimes;
N(:,2:end) = waveforms;
csvwrite(fileName_waveform,N);
A.1.3 Visual Feed Back Mode In Phantom

%%% Visual Feedback Mode In-Phantom
%%% Modified based on PicoScope 2000 Series Instrument Driver Oscilloscope
%%% Block Data Capture with Simple Trigger Example
%%% *Copyright:* 2013 - 2017 Pico Technology Ltd. All rights reserved.

%%% Setup PicoScope:

%%% The PicoScope 2205A uses the API functions from the ps2000 shared library and so requires the PicoScope 2000 Series MATLAB Generic Instrument Driver. Please install this using the Add-Ons Explorer menu item in the toolbar of the MATLAB Home tab.

%%% Please note that there is an error in the PS2000Config file which affects path ordering - please ensure that you add the win64 folder from the Instrument Driver root folder to the top of the MATLAB path. This will be fixed in due course.

%%% Also, please copy the dlls from this folder(https://drive.google.com/drive/folders/0Bz9oJyp18PmZR2N2WThWYXVnVzQ?usp=sharing) or (https://drive.google.com/open?id=OBz9oJyp18PmZR2N2WThWYXVnVzQ) to the following location:

%%% C:\Program Files\Pico Technology\SDK\lib

%%% Setup Arduino: MATLAB Support Package for Arduino Hardware version 17.1.1

%%% Declaration: this version may not be the exact code version used in gathering the
%%% data shown in Chapter 5. In particular, the threshold may vary based
%%% on setup etc.

%%% License: MIT License, No commercial use

%%% Clear Command Window and Close any Figures

clc
close all
clear all
%% Load Configuration Information
PS2000Config;

%% Device Connection

%% Create a device object.
ps2000DeviceObj = icdevice('picotech_ps2000_generic.mdd');

%% Connect device object to hardware.
connect(ps2000DeviceObj);

%% Arduino Connection
a = arduino('com10','nano','Libraries','I2C');
tmp102 = i2cdev(a, '0x06');

%% Obtain Device Groups

%% Obtain references to device groups to access their respective
 properties
 and functions.

%% Block specific properties and functions are located in the Instrument
 Driver's Block group.
blockGroupObj = get(ps2000DeviceObj, 'Block');
blockGroupObj = blockGroupObj(1);

%% Trigger specific properties and functions are located in the
 Instrument
 Driver's Trigger group.
triggerGroupObj = get(ps2000DeviceObj, 'Trigger');
triggerGroupObj = triggerGroupObj(1);

%% Configure the Device

%% Enable Channels A and B, set the sampling interval, number of samples
to
% collect and set a simple trigger.

% Set channels:

% Channel : 0 (ps2000Enuminfo.enPS2000Channel.PS2000-CHANNEL-A)
% Enabled : 1 (PicoConstants.TRUE)
% DC : 1 (DC Coupling)
% Range : 6 (ps2000Enuminfo.enPS2000Range.PS2000.1V)
%[status.setChA] = invoke(ps2000DeviceObj, 'ps2000SetChannel', 0, 1, 1, 6);
% Range : 8 (ps2000Enuminfo.enPS2000Range.PS2000.5V)
%[status.setChA] = invoke(ps2000DeviceObj, 'ps2000SetChannel', 0, 1, 1, 8);

% Channel : 1 (ps2000Enuminfo.enPS2000Channel.PS2000-CHANNEL-B)
% Enabled : 1 (PicoConstants.TRUE)
% DC : 1 (DC Coupling)
% Range : 7 (ps2000Enuminfo.enPS2000Range.PS2000.2V)
% [status.setChB] = invoke(ps2000DeviceObj, 'ps2000SetChannel', 1, 1, 1, 7);
% Range : 8 (ps2000Enuminfo.enPS2000Range.PS2000.5V)
% [status.setChB] = invoke(ps2000DeviceObj, 'ps2000SetChannel', 1, 1, 1, 8);

% Set sampling interval and number of samples to collect:

% Set sampling interval - the actual sampling interval used by the
driver
% will be returned.

% timeIntervalUs : 100 (microseconds) --> 10us / 0.01 ms --> 100us / 0.1
ms
[samplingIntervalUs, maxBlockSamples] = invoke(blockGroupObj, '
% Confirm the timebase index selected.
timebaseIndex = get(blockGroupObj, 'timebase');

% Set the number of samples to collect. 2048 -->
set(ps2000DeviceObj, 'numberOfSamples', 5004);
%
% Set simple trigger:

% Set the trigger delay to 50% (samples will be shown either side of the
% trigger point. --> Will show 10% left
set(triggerGroupObj, 'delay', -10.0);

% Set the autoTriggerMs property in order to automatically trigger the
% oscilloscope after 1 second if a trigger event has not occurred. Set
to 0
% to wait indefinitely for a trigger event. --> 1000 for 1 second
set(triggerGroupObj, 'autoTriggerMs', 0);

% Parameters taken from config file loaded into workspace.

[simpleTriggerStatus] = invoke(triggerGroupObj, 'setSimpleTrigger', ...
        simpleTrigger.threshold, ... 
    ps2000ConfigInfo.simpleTrigger.direction);

%% get one reading from sensor
data = read(tmp102, 6);
frameIndex = typecast([data(2) data(1)], 'uint16');
timeStamp = typecast([data(4) data(3)], 'uint16'); % 0.1ms increments
output = typecast([data(6) data(5)], 'uint16');
clc

disp(dec2hex(data));
disp(strcat('frame index: ', num2str(frameIndex)));  
disp(strcat('time stamp: ', num2str(timeStamp)));  
disp(strcat('output: ', num2str(output)));  

%% Update baseline <-- this doesn't seem to do anything  
baselineMSB = data(5);  
baselineLSB = data(6);  
% baseline = typecast([data(6) data(5)],'uint16');  

updateBaseline = [uint8(2),... % '0x02'  
    uint8(41),... % write offset in register block  
    uint8(2),... % number of bytes to write  
    baselineMSB, ... %data to write  
    baselineLSB, ... %data to write  
    uint8(hex2dec('FF'))] %0xFF - end of packet delimiter  
];  

write(tmpl02, updateBaseline); %calibrate to baseline  

pause(1);  

%% Data Collection  
data = read(tmpl02, 6);  
baseline = typecast([data(6) data(5)],'uint16');  

trialnum = 100;  
depths = zeros(1,trialnum);  
force = zeros(size(depths));  
x = zeros(size(depths));  
y = zeros(size(depths));  
z = zeros(size(depths));  

time = zeros(size(depths));  

% get one set of data  
[bufferTimes, bufferChA, bufferChB, numDataValues, timeIndisposedMs] =  
    invoke(blockGroupObj, 'getBlockData');
stopStatus = invoke(ps2000DeviceObj, 'ps2000Stop'); % Stop the Device
bufferTimes = double(bufferTimes);

timesUnits = timeunits(get(blockGroupObj, 'timeUnits')); % Find the time units used by the driver.
timeLabel = strcat('Time (', timesUnits, ',

waveforms = zeros(length(bufferTimes), trialnum);
detectarr = zeros(size(bufferTimes));
T = bufferTimes(2) - bufferTimes(1); %ps
T = T*le-6; %us
fs = 1/T; %MHz
b = firl(128, [0.9 1.1]*2/fs); %for single and probe

speedOfSound = 1000*86*2/170.7969; %calibrated on s9 fitted time, but any should produce the same result (Day 12 notes)
dis = bufferTimes*le-6*speedOfSound/1000; %mm
dis = dis/2;
disthresh = 12;
thresh = 0.6 * 28;

edges = [];
tic; %start timer

i=1;
figure(1);
disp('starting data collection...

while i <= trialnum

% Capture a block of data on Channels A and B together with times. Data for channels is returned in millivolts.
[bufferTimes, waveforms(:,i), ~, numDataValues, timeIndisposedMs] = invoke(blockGroupObj, 'getBlockData');

90
%% Stop the Device
stopStatus = invoke(ps2000DeviceObj, 'ps2000Stop');

%% Process Data

% Filter
bufferChA_filt = filtfilt(b, 1, waveforms(:, i));

% abstract envelope
[yupper, ~] = envelope(bufferChA_filt);

[.., locs] = findpeaks(yupper, 'MinPeakHeight', thresh);

if isempty(locs)
    continue
else
    if dis(locs(l)) < 10 && dis(locs(1)) > 0 % if triggered properly
        yupper(dis<disthresh) = 0;
    end
    % threshold
detects = zeros(size(bufferTimes));
detectlocs = find(yupper > thresh);

    % find peaks
[pks, pklocs] = findpeaks(yupper, 'MinPeakHeight', thresh);
disp(dis(locs));

    if isempty(pklocs)
    continue
    elseif dis(pklocs(l)) > 90
    continue
    else
    depths(i) = dis(pklocs(l));
    end
detects(detectlocs) = 1;
detect_arr = detect_arr + detects;

edges = [edges dis(pklocs)'];

% record force
data = read(tmp102,6);
force(i) = typecast([data(6) data(5)],'uint16');

% Read accelerometer
x(i) = readVoltage(a,'A0');
y(i) = readVoltage(a,'A1');
z(i) = readVoltage(a,'A2');

edges = [edges dis(locs)'];

disp(num2str(depths(i)));
disp(num2str(i));

plot(dis, bufferChA_filt);
hold on;
plot(dis,detects*thresh, 'LineWidth',3);
my_vline(depths(i),'r');
xlabel('dis [mm]');
ylim([-1000 1000]);
grid on
hold off

i = i + 1;
end
end

pause(1);

disp(i);
end

force = force - double(baseline);
force = force * 4; \% calibrated by the 200g calibration weight, unit: gram

%% Disconnect
% Disconnect device object from hardware.

disconnect(ps2000DeviceObj);
delete(ps2000DeviceObj);
%% Plot depths, force, and acceleration.
figure;
subplot 311
plot(time, depths,'o-');
title('depth [mm]');
xlabel('time [sec]');
grid on;
grid minor
ylim([36 47])
\% ylim([14 27])

subplot 312
plot(time, force,'x-');
title('force [g]');
xlabel('time [sec]')
grid on
grid minor

subplot 313
plot(time,x,time,y,time,z);
legend('x','y','z');
xlabel('time [sec]');
title('accelerometer reading');
grid on;
grid minor

%% Plot the last Waveform (to visually confirm that valid waveforms have
been acquired)

figure;

plot( dis, bufferChA.filt, dis, yupper, dis, detects, 'LineWidth', 1);
hold on
my.vline(depths(l-1), 'r');
grid on
title('Block Data Acquisition');
xlabel('distance []');

%% Save time, depths, force, and acceleration data to CSV
clc
curr.time = datetime('now');
curr.time.str = datestr(curr.time);
curr.time.str = strrep(curr.time.str, ' ', '-');
curr.time.str = strrep(curr.time.str, ':', '-');
curr.time.str = strrep(curr.time.str, '-', '-');
timeDepthsForceAccel = strcat(curr.time.str, '.csv');
waveform.filename = strcat(curr.time.str, '-waveforms.csv');

pathName='C:\Users\simulator II\Documents\MATLAB\Staircase\16';
fileName=fullfile(pathName, timeDepthsForceAccel);
disp(fileName)

fileName.waveform=fullfile(pathName, waveform.filename);
disp(fileName.waveform)

M = zeros(length(time), 6); % time, depths, force, x, y, z
M(:,1) = time;
M(:,2) = depths;
M(:,3) = force;
M(:,4) = x;
M(:,5) = y;
M(:,6) = z;
csvwrite(fileName, M);

[r, c] = size(waveforms);
N = zeros(r, c+1);
N(:,1) = bufferTimes;
N(:,2:end) = waveforms;
csvwrite(fileName.waveform,N);

% save the image
img_name = strcat(curr.time.str, '.png');
img_name = fullfile(pathName,img_name);
saveas(figure(1),img_name)
A.1.4 Visual Feedback Mode *in-vivo*

%% Visual Feedback Mode In-vivo
% Modified based on PicoScope 2000 Series Instrument Driver Oscilloscope
Block Data Capture with Simple Trigger Example
% *Copyright:* 2013 - 2017 Pico Technology Ltd. All rights reserved.

% Setup PicoScope:

% The PicoScope 2205A uses the API functions from the ps2000 shared
library and so requires the PicoScope 2000 Series MATLAB Generic
Instrument Driver. Please install this using the Add-Ons Explorer
menu item in the toolbar of the MATLAB Home tab.
% Please note that there is an error in the PS2000Config file which
affects path ordering - please ensure that you add the win64 folder
from the Instrument Driver root folder to the top of the MATLAB path.
This will be fixed in due course.
% Also, please copy the dlls from this folder(https://drive.google.com/
drive/folders/0Bz9oJyplSPmZR2N2WThWYXVnVzQ?usp=sharing) or (https://
drive.google.com/open?id=0Bz9oJyp18PmZR2N2WThWYXVnVzQ) to the
following location:
% C:\Program Files\Pico Technology\SDK\lib

% Setup Arduino: MATLAB Support Package for Arduino Hardware version
17.1.1

% Declaration: this version may not be the exact code version used in
gathering the
% data shown in Chapter 5. In particular, the threshold may vary based
% on setup etc.

% License: MIT License, No commercial use

%% Clear Command Window and Close any Figures
try
disconnect(ps2000DeviceObj);
delete(ps2000DeviceObj);
catch
    %do nothing
end
clc;
close all;
clear all;
% Load Configuration Information
PS2000Config;

%% Device Connection

% Create a device object.
ps2000DeviceObj = icdevice('picotech.ps2000.generic.mdd');

% Connect device object to hardware.
connect(ps2000DeviceObj);

%% Arduino Connection

% a = arduino('com8','uno','Libraries','I2C');
a = arduino('com10','nano','Libraries','I2C');
tmpl02 = i2cdev(a, '0x06');

%% Obtain Device Groups
% Obtain references to device groups to access their respective properties
% and functions.
% Block specific properties and functions are located in the Instrument Driver's Block group.

blockGroupObj = get(ps2000DeviceObj, 'Block');
blockGroupObj = blockGroupObj(1);

% Trigger specific properties and functions are located in the
Instrument
% Driver's Trigger group.

triggerGroupObj = get(ps2000DeviceObj, 'Trigger');
triggerGroupObj = triggerGroupObj(1);

%% Configure the Device
% Enable Channels A and B, set the sampling interval, number of samples
to
% collect and set a simple trigger.

%%
% Set channels:

% Channel    : 0 (ps2000Enuminfo.enPS2000Channel.PS2000.CHANNEL.A)
% Enabled    : 1 (PicoConstants.TRUE)
% DC         : 1 (DC Coupling)
% Range      : 6 (ps2000Enuminfo.enPS2000Range.PS2000.1V)
[status.setChA] = invoke(ps2000DeviceObj, 'ps2000SetChannel', 0, 1, 1,
                         6);
% Range      : 8 (ps2000Enuminfo.enPS2000Range.PS2000.5V)
[status.setChA] = invoke(ps2000DeviceObj, 'ps2000SetChannel', 0, 1, 1,
                         8);

% Channel    : 1 (ps2000Enuminfo.enPS2000Channel.PS2000.CHANNEL.B)
% Enabled    : 1 (PicoConstants.TRUE)
% DC         : 1 (DC Coupling)
% Range      : 7 (ps2000Enuminfo.enPS2000Range.PS2000.2V)
% [status.setChB] = invoke(ps2000DeviceObj, 'ps2000SetChannel', 1, 1, 1,
                         7);
% Range      : 8 (ps2000Enuminfo.enPS2000Range.PS2000.5V)
[status.setChB] = invoke(ps2000DeviceObj, 'ps2000SetChannel', 1, 1, 1,
                         8);
% Set sampling interval and number of samples to collect:

% Set sampling interval - the actual sampling interval used by the
driver
% will be returned.

% timeIntervalUs : 100 (microseconds) --> 10us / 0.01 ms --> 100us / 0.1
ms
[samplingIntervalUs, maxBlockSamples] = invoke(blockGroupObj, 'setBlockIntervalUs', 0.1);

% Confirm the timebase index selected.
timebaseIndex = get(blockGroupObj, 'timebase');

% Set the number of samples to collect. 2048 -->
set(ps2000DeviceObj, 'numberOfSamples', 5004); %3072 --> -3.061 - 27.65
ms

% Set simple trigger:

% Set the trigger delay to 50% (samples will be shown either side of the
% trigger point. ----> Will show 10% left
set(triggerGroupObj, 'delay', -10.0);

% Set the autoTriggerMs property in order to automatically trigger the
% oscilloscope after 1 second if a trigger event has not occurred. Set
to 0
% to wait indefinitely for a trigger event. ----> 1000 for 1 second
set(triggerGroupObj, 'autoTriggerMs', 0);

% Parameters taken from config file loaded into workspace.

[simpleTriggerStatus] = invoke(triggerGroupObj, 'setSimpleTrigger', ...
simpleTrigger.threshold, ...
ps2000ConfigInfo.simpleTrigger.direction);

%% get one reading from sensor
data = read(tmpl02, 6);
frameIndex = typecast([data(2) data(1)],'uint16');
timeInterval = typecast([data(4) data(3)],'uint16'); % 0.1ms increments
output = typecast([data(6) data(5)],'uint16');
clc
disp(dec2hex(data));
disp(strcat('frame index: ', num2str(frameIndex)));
disp(strcat('time stamp: ', num2str(timeStamp)));
disp(strcat('output: ', num2str(output)));

%% Update baseline <== this doesn't seem to do anything
baselineMSB = data(5);
baselineLSB = data(6);
% baseline = typecast([data(6) data(5)],'uint16');
updateBaseline = [uint8(2),...
uint8(41),... % write offset in register block
uint8(2),... % number of bytes to write
baselineMSB, ... % data to write
baselineLSB, ... % data to write
uint8(hex2dec('FF')) % 0xFF - end of packet delimiter
];
write(tmpl02, updateBaseline); % calibrate to baseline

pause(1);

%% Data Collection
data = read(tmpl02, 6);
baseline = typecast([data(6) data(5)],'uint16');
trialnum = 50;
depths = zeros(1,trialnum);
force = zeros(size(depths));
x = zeros(size(depths));
y = zeros(size(depths));
z = zeros(size(depths));
time = zeros(size(depths));

% get one set of data
[bufferTimes, bufferChA, bufferChB, numDataValues, timeIndisposedMs] =
    invoke(blockGroupObj, 'getBlockData');
stopStatus = invoke(ps2000DeviceObj, 'ps2000Stop'); % Stop the Device
bufferTimes = double(bufferTimes);

timesUnits = timeunits(get(blockGroupObj, 'timeUnits')); % Find the time
    units used by the driver.
timeLabel = strcat('Time (' , timesUnits, ')');

waveforms = zeros(length(bufferTimes), trialnum);
detect_arr = zeros(size(bufferTimes));
T = bufferTimes(2) - bufferTimes(1); % ps
T = T*1e-6; % us
fs = 1/T; % MHz
b = fir1(128, [0.9 1.1]*2/fs); % for single and probe

speedOfSound = 1540; % m/s
dis = bufferTimes*1e-6*speedOfSound/1000; % mm
dis = dis/2;
disthresh = 20;
disthresh_short = 10;
thresh = 100;
peak_thresh = 300;
force_thresh = 40;
edges = [];}
figure(1);  
i = 1;  
tic; % start time  
disp('starting data collection...');  
while i <= trialnum

% Capture a block of data on Channels A and B together with times.  
% Data for channels is returned in millivolts.  
[bufferTimes, waveforms(:,i), -, numDataValues, timeIndisposedMs] =  
invoke(blockGroupObj, 'getBlockData');

%% Stop the Device  
stopStatus = invoke(ps2000DeviceObj, 'ps2000Stop');

%% Process Data  
% Filter  
bufferChA_filt = filtfilt(b,1,waveforms(:,i));

% abstract envelope  
[yupper,-] = envelope(bufferChA_filt);  
pRMS = rms(yupper(dis>= disthresh))^2; % signal power beyond  
distribresh  
if pRMS < 1000  
    yupper(dis<disthresh.short) = 0;  
    disp('Short Distance...');
else  
    yupper(dis<disthresh) = 0;
end

% threshold  
detects = zeros(size(bufferTimes));  
detectloc = find(yupper > thresh);
find peaks
[pks, locs] = findpeaks(yupper,'MinPeakHeight', peak.thresh);
disp(dis(locs));

if isempty(detectlocs) && isempty(locs)
    depths(i) = 0;
else if isempty(locs)
    depths(i) = dis(detectlocs(1));
    detects(detectlocs) = 1;
    detect_arr = detect_arr + detects;
    edges = [edges dis(locs)'];
else if dis(locs(1)) > 120
    depths(i) = 0;
else
    depths(i) = dis(locs(1));
    detects(detectlocs) = 1;
    detect_arr = detect_arr + detects;
    edges = [edges dis(locs)'];
    time(i) = toc;

record force
data = read(tmpl02,6);
force(i) = typecast([data(6) data(5)],'uint16');

Read accelerometer
x(i) = readVoltage(a,'A0');
y(i) = readVoltage(a,'A1');
z(i) = readVoltage(a,'A2');

edges = [edges dis(locs)'];
disp(num2str(depths(i)));
disp(num2str(i));
plot(dis, bufferChAfilt);
hold on;
plot(dis,detect*thresh,'LineWidth',3);
my_vline(depths(i),'r');
xlabel('dis [mm]');
ylim([-1000 1000]);
grid on
hold off

i = i + 1;

end

pause(1);

end

force = force - double(baseline);

%%
figure(2);
subplot 411
plot(time, depths,'o-');
hold on;
plot(time, force,'x-');
legend('depths', 'force')
title('depth | force');
grid on;
grid minor;

subplot 412
plot(dis, detect.arr);
xlabel('distance [mm]')
grid on
grid minor;
title('detect/above threshold')

subplot 413
hist(edges,max(ceil(edges)))
grid on
grid minor
title('histogram of above threshold rising edge');

subplot 414
plot(time,x,time,y,time,z);
legend('x','y','z');
xlabel('time [sec]');
title('accelerometer reading');
grid on;
grid minor

%% Disconnect
% Disconnect device object from hardware.

disconnect(ps2000DeviceObj);
delete(ps2000DeviceObj);

%% Save Data

clc
curr_time = datetime('now');
curr_time_str = datestr(curr_time);
curr_time_str = strrep(curr_time_str, ' ', '_');
curr_time_str = strrep(curr_time_str, ':', '_');
curr_time_str = strrep(curr_time_str, '-', '_');
timeDepthsForceAccel = strcat(curr_time_str, '.csv');
waveform_filename = strcat(curr_time_str,'.waveforms.csv');
edge_filename = strcat(curr_time_str.'_edge.csv');

pathName='C:\Users\simulator II\Documents\MATLAB\Data collection\Leg
Indentation Test Through'

fileName = fullfile(pathName, timeDepthsForceAccel);
disp(fileName)

fileName.edge = fullfile(pathName, edge_filename);
disp(edge_filename)

fileName.waveform = fullfile(pathName, waveform_filename);
disp(fileName.waveform)

csvwrite(fileName.edge, edges');

M = zeros(length(time), 6); % time, depths, force, x, y, z
M(:, 1) = time;
M(:, 2) = depths;
M(:, 3) = force;
M(:, 4) = x;
M(:, 5) = y;
M(:, 6) = z;
csvwrite(fileName, M);

[r, c] = size(waveforms);
N = zeros(r, c+2);
N(:, 1) = bufferTimes;
N(:, 2:end-1) = waveforms;
N(:, end) = detect_arr;
csvwrite(fileName.waveform, N);

% save the image
img.name = strcat(curr.time_str, '.fig');
img.name = fullfile(pathName, img.name);
saveas(figure(2), img.name)
A.2 Post Processing

A.2.1 Post Processing of Depth Sensing in Phantom

```matlab
% Post Processing of saved waveforms for depth sensing in Phantom
% zixiliu@mit.edu
%
% Declaration: this version is for depth sensing in phantom as described
% in Chapter 5.

% License: MIT License, No commercial use
%
clear
close all
clc

% Constants
font_size = 14;
font_weight = 'bold';

% open the newest file
spotnum = 76;
waveform_filenames = dir(fullfile(strcat(num2str(spotnum),'/','*.wav')));
time_filenames = dir(fullfile(strcat(num2str(spotnum),'/','*.timestamp.csv')));

figure;
for j = 1:length(waveform_filenames)

  waveform_filename = strcat(num2str(spotnum),'/',waveform_filenames(j).name);
time_filename = strcat(num2str(spotnum),'/',time_filenames(j).name);
```

107
disp(waveform_filename)
disp(timestamp_filename)

waveforms = csvread(waveform_filename);
time= csvread(timestamp_filename);

bufferTimes = waveforms(:,1);
waveforms = waveforms(:,2:end);
[-,trialnum] = size(waveforms);

%% set up for precessing
depths = zeros(1,trialnum);
detect_arr = zeros(size(bufferTimes));
T = bufferTimes(2) - bufferTimes(1); %ps
T = T*1e-6; %us
fs = 1/T; %MHz
b = fir1(128,[0.9 1.1]*2/fs); %for single and probe

speedOfSound = 1000*86*2/170.7969; %calibrated on s9 fitted time,
    but any should produce the same result (Day 12 notes)
dis = bufferTimes*1e-6*speedOfSound/1000; %mm
dis = dis/2;
disthresh = 12;
thresh = 0.6 * 28;

edges = [];

%% process data
warning off

for i = 1:trialnum
    % Filter
    bufferChA_filt = filtfilt(b,1,waveforms(:,i));
    
    % abstract envelope

108
[yupper, -] = envelope(bufferChA.filt);

[-, locs] = findpeaks(yupper, 'MinPeakHeight', thresh);

if isempty(locs)
    continue
else
    if dis(locs(1)) < 10 & dis(locs(1)) > 0  // if triggered properly
        yupper(dis<disthresh) = 0;

        % threshold
        detects = zeros(size(bufferTimes));
        detectlocs = find(yupper > thresh);

        % find peaks
        [pks, pklocs] = findpeaks(yupper, 'MinPeakHeight', thresh);
        disp(dis(locs));

        if isempty(pklocs)
            depths(i) = 0;
        elseif dis(pklocs(1)) > 90
            continue
        else
            depths(i) = dis(pklocs(1));
            detects(detectlocs) = 1;
            detect_arr = detect_arr + detects;

            edges = [edges dis(pklocs)'];
        end
    end
end
end
end

%% plotting
figure;
hist(edges,100)

end
A.2.2 Post Processing of Depth Sensing in-Vivo

%% Post Processing of saved waveforms for bone depth sensing in-vivo
%% zixiliu@mit.edu
%%
%% Declaration: this version is for depth sensing in phantom as described
%% in Chapter 5.

%% License: MIT License, No commercial use
%%
clear
close all
clc

%% Constants
font_size = 14;
font_weight = 'bold';

%% open the newest file
spotnum = 4;

waveform_filenames = dir(fullfile(strcat(num2str(spotnum), '/','*w
waveforms.csv')));
time_filenames = dir(fullfile(strcat(num2str(spotnum), '/','*timestamp.
csv')));

h = figure;
for j = 1: length(waveform_filenames)

waveform_filename = strcat(num2str(spotnum), '/', waveform_filenames(j).
.name);

time_filename = strcat(num2str(spotnum), '/', time_filenames(j).name);

111
disp(waveform_filename)
disp(time_filename)

M = csvread(waveform_filename);
time = csvread(time_filename);

bufferTimes = M(:,1);
waveforms = M(:,2:end);
[-,trialnum] = size(waveforms);

%% set up for processing
detect_arr = zeros(size(bufferTimes));
T = bufferTimes(2) - bufferTimes(1); %ps
T = T * 1e-6; %us
fs = 1/T; %MHz
b = fir1(128, [0.9 1.1] * 2/fs); %for single and probe

speedOfSound = 1540;
dis = bufferTimes * 1e-6 * speedOfSound / 1000; %mm
dis = dis / 2;
disthresh = 20;
disthresh_short = 10;
thresh = 100;
peak_thresh = 300;
force_thresh = 40;

edges = [];

depths = zeros(trialnum, 1);

%% process data

warning off

for i = 1:trialnum

    % Filter
bufferChA filt = filtfilt(b,1, waveforms(:,i));

% abstract envelope
[yupper,~] = envelope(bufferChA filt);
pRMS = rms(yupper(dis >= disthresh))^2; % signal power beyond disthresh

if pRMS < 1000
    yupper(dis<disthresh_short) = 0;
else
    yupper(dis<disthresh) = 0;
end

% threshold
detects = zeros(size(bufferTimes));
detectlocs = find(yupper > thresh);

% find peaks
[pks, locs] = findpeaks(yupper, 'MinPeakHeight', peak_thresh);

if isempty(detectlocs) && isempty(locs)
    depths(i) = 0;
elseif isempty(locs)
    depths(i) = dis(detectlocs(1));
detects(detectlocs) = 1;
detect_arr = detect_arr + detects;
detects(detectlocs) = 1;
detect_arr = detect_arr + detects;
detects(detectlocs) = 1;
detect_arr = detect_arr + detects;
detects(detectlocs) = 1;
detect_arr = detect_arr + detects;
detects(detectlocs) = 1;
detect_arr = detect_arr + detects;
end

if isempty(locs(1)) > 120
    depths(i) = 0;
else
    depths(i) = dis(locs(1));
detects(detectlocs) = 1;
detect_arr = detect_arr + detects;
edges = [edges dis(locs)'];

if isempty(locs(1)) > 120
    depths(i) = 0;
else
    depths(i) = dis(locs(1));
detects(detectlocs) = 1;
detect_arr = detect_arr + detects;
edges = [edges dis(locs)'];
end

113
% subplot 211
% plot(bufferTimes*le-9,waveforms(:,i)/1000);
% xlabel('Time [ms]', 'FontSize', font.size, 'FontWeight',
% font.weight);
% ylabel('Value [mV]', 'FontSize', font.size, 'FontWeight',
% font.weight);
% title('Raw Signal', 'FontSize', font.size, 'FontWeight',
% font.weight);
% grid on
%
% subplot 212
% plot(dis, yupper/1000, dis, peak.thresh.detects/1000);
% hold on
% ylim([0 1]);
% legend('Filtered Upper Envelope', 'Above threshold Detection');
% xlabel('Distance [mm]', 'FontSize', font.size, 'FontWeight',
% font.weight);
% ylabel('Value [mV]', 'FontSize', font.size, 'FontWeight',
% font.weight);
% title('Processed Signal', 'FontSize', font.size, 'FontWeight',
% font.weight);
% grid on
%
% for ii = 1:length(locs)
%   my.vline(dis(locs(ii)),'r');
% end
% hold off
%
% waitforbuttonpress
end

warning on

%%%%%%%% plotting
figure;
subplot 311
plot(time, depths,'o-');
xlabel('time (sec)')
ylim([0 110])
title('depth');
grid on;
grid minor;

subplot 312
plot(dis, detect_arr);
xlabel('distance [mm]')
grid on
grid minor;
title('detect/above threshold')
xlim([0 110])

subplot 313
h = histogram(edges);
h.Normalization = 'probability';
h.NumBins = 200;
% h.FaceColor = 'b';
grid on
grid minor

title('histogram of above threshold rising edge');
% ylim([0 110])
xlim([0 110])
end
A.2.3 Post Processing of Indentation Sensing in Phantom

%% Post Processing of \textit{saved} waveforms
\% zixiliu@mit.edu
\% License: MIT License, No commercial use
\% Constants
font.size = 14;
font.weight = 'bold';

%% open file
spotnum = 76;
spotname = '76';

waveform_filenames = dir(fullfile(strcat(num2str(spotname), '/','*
    waveform.csv')));

h = figure;
for j = 1: length(waveform_filenames)
    waveform.filename = strcat(num2str(spotname), '/',waveform_filenames(j
        ).name);
    time.filename = strcat(num2str(spotname), '/',waveform_filenames(j).
        name(1:end-14),'.csv');

    disp(waveform.filename)
    disp(time.filename)

    M = csvread(waveform.filename);
    N = csvread(time.filename);
    time = N(:,1);

    bufferTimes = M(:,1);
    waveforms = M(:,2:end);
    [-,trialnum] = size(waveforms);

%%% set up for processing
detect_arr = zeros(size(bufferTimes));
T = bufferTimes(2) - bufferTimes(1);  %ps
T = T*1e-6;  %us
fs = 1/T;  %MHz
b = fir1(128, [0.9 1.1]*2/fs);  %for single and probe

speedOfSound = 1007;
dis = bufferTimes*1e-6*speedOfSound/1000;  %mm
dis = dis/2;
disthresh = 20;
disthresh_short = 10;
thresh = 100;

depths = zeros(trialnum, 1);

for i = 1:trialnum
    % Filter
    bufferChA_filt = filtfilt(b,1,waveforms(:,i));

    % Abstract envelope
    [yupper,-] = envelope(bufferChA_filt);
pRMS = rms(yupper(dis>= disthresh)).^2;  %signal power beyond disthresh

    if pRMS < 1000
        yupper(dis<disthresh_short) = 0;
    else
        yupper(dis<disthresh) = 0;
    end

    % Threshold
    detects = zeros(size(bufferTimes));
detectlocs = find(yupper > thresh);
% find peaks
[pks, locs] = findpeaks(yupper, 'MinPeakHeight', thresh);

if isempty(detectlocs) && isempty(locs)
    depths(i) = 0;
else if isempty(locs)
    disp('ahhh!');
    depths(i) = dis(detectlocs(1));
    detects(detectlocs) = 1;
else if dis(locs(1)) > 90
    depths(i) = 0;
else
    detects(detectlocs) = 1;
    detected_depths = dis(locs);
    this_depth = detected_depths(detected_depths < (spotnum) & detected_depths > (spotnum-10));

    if isempty(this_depth)
        if i == 1
            depths(i) = 0;
        else
            depths(i) = depths(i-1);
        end
    elseif length(this_depth) == 1
        depths(i) = this_depth;
    else

        if i == 1
            depths(i) = this_depth(1);
        else
            tmp = abs(this_depth - depths(i-1));
            [~, idx] = min(tmp);
            depths(i) = this_depth(idx);

118
%% plotting
fig = figure;
plot(time, depths,'o-');
xlabel('time (sec)')
ylim([spotnum-10 spotnum+2])
title(strcat('Depths in Trail in Step No.',num2str(spotnum)));
grid on;
grid minor;
set(gca, 'fontsize',font.size,'fontweight',font.weight);

end
%% Save figure
saveas(fig,strcat('step',num2str(spotname),'.fig'))

%% Save .mat
M = zeros(length(time),2);
M(:,1) = time;
M(:,2) = depths;

save([spotname,'result.mat'],M);
A.2.4 Post Processing of Indentation Sensing in-Vivo

```matlab
%% Post Processing of saved waveforms
% zixiliu@mit.edu
%
% License: MIT License, No commercial use
%%
close all
clear
clc
%% constants
speedOfSound = 1540;
disthresh = 20;
disthresh.short = 10;
thresh = 300;
trialnum = 50;

font_size = 14;
line_width = 2;

%%
filenames = dir('*.waveforms.csv');

for j = 1:length(filenames)
    % load data
    filename.prefix = filenames(j).name(1:end-14);
    disp(filename.prefix)
    N = csvread(filenames(j).name);
    M = csvread(strcat(filename.prefix, '.csv'));

    time = M(:,1);
    force = M(:,3); force = force - min(force);
    x = M(:,4);
    y = M(:,5);
    z = M(:,6);
```
bufferTimes = N(:,1);
waveforms = N(:,2:end-1);

T = bufferTimes(2) - bufferTimes(1); % ps
T = T*1e-6; % us
fs = 1/T; % MHz
b = fir1(128,[0.9 1.1]*2/fs); % for single and probe
dis = bufferTimes*1e-6*speedOfSound/1000; % mm
dis = dis/2;
depths = zeros(1,trialnum);

%% Process Data
for i = 1:trialnum
    % Filter
    bufferChA.filt = filtfilt(b,1,waveforms(:,i));

    % abstract envelope
    [yupper,~] = envelope(bufferChA.filt);
pRMS = rms(yupper(dis>= disthresh))^2; % signal power beyond disthresh
    if pRMS < 1000
        yupper(dis<disthresh_short) = 0;
    else
        yupper(dis<disthresh) = 0;
    end

    % threshold
    detects = zeros(size(bufferTimes));
detectlocs = find(yupper > thresh);

    % find peaks
    [pks, locs] = findpeaks(yupper,'MinPeakHeight', thresh);
    if isempty(locs)
        depths(i) = 0;
    end
end
elseif dis(detectlocs(l)) > 120
    depths(i) = 0;
else
    detects(detectlocs) = 1;
    detected_depths = dis(locs);
    this_depth = detected_depths(detected_depths < 101 &
                              detected_depths > 85); % The thickness of the limb in
    this case study is roughly between 85 and 101
    if isempty(this_depth)
        if i == 1
            depths(i) = 0;
        else
            depths(i) = depths(i-1);
        end
        elseif length(this_depth) == 1
            depths(i) = this_depth;
        else
            if i == 1
                depths(i) = this_depth(end);
            else
                tmp = abs(this_depth-depths(i-1));
                [~, idx] = min(tmp);
                depths(i) = this_depth(idx);
            end
        end
    end
end

% calculate tilt angle
beta = atan2(y, sqrt(x.^2 + z.^2));

figure;
subplot 211
title_name = strcat('In-vivo Indentation Experiment trial', num2str(j)
indentation, force vs. time); title(title.name,'FontSize',font.size); grid on; grid minor;

yyaxis left
plot(time, max(depths) - depths,'o-','LineWidth',line.width);
ylabel('Indentation [mm]')
ylim([0 12])

yyaxis right
plot(time, force*0.0098,'x-','LineWidth',line.width);
ylabel('Force [N]')

legend('depths','force')

subplot 212
plot(time,beta,'LineWidth',line.width);
xlabel('time [sec]','FontSize',font.size);
title('Tilt Angle [radian]');
grid on;
grid minor
xlim([5 50])

waitforbuttonpress

end
A.3 MRI Image Processing

The author of all code in this section is Kevin Moerman, kmoerman@mit.edu.

A.3.1 Import DICOM Data

%% This file uses Gibbon for plotting, https://www.gibboncode.org/,
   Gibbon license: GNU General Public License v3.0
%% License: MIT License, No commercial use
%% author: Kevin Moerman, kmoerman@mit.edu
%% dcmFolder2MATobject
%% Below is a demonstration of the features of the |dcmFolder2MATobject| function

%% Syntax
%% |dcmFolder2MATobject(PathName,MaxVarSize)|

%% Description
%% The |dcmFolder2MATobject| function converts DICOM data to a MATLAB mat (object and
%% or) file.

%% Examples

clear; close all; clc;

%% Plot settings

%% Example: CONVERTING DICOM IMAGE DATA TO A MAT OBJECT
%% Below some example code is shown to convert all DICOM files inside a
%% folder (including its subfolders) to the IMDAT format. The function
%% |dcmFolder2MATobject| converts the DICOM data to a matlab MAT object and
% saves it under the name IMDAT.mat inside a subfolder called IMDAT.
% A waitbar appears showing the process of the data conversion for the
% DICOM information and image data. Multiple types of image data (e.g.
% phase, real, imaginary, magnitude data) is stored separately. Also
% several DICOM info fields are harvested and stored.

% The IMDAT.mat object contains the following:

% IMDAT-struct =

%                             G: [1x1 struct] %The geometry parameters
%                                 ImageSize: [128 128 17 20] % The image size
%                                 ImageTypesUni: {'ORIGINAL\PRIMARY\M.FFE\M\FFE'} % The image type
% or types
%                                 type.l: [4-D uint16] % The image data matrix
%                                 type.l.info: [1x340 struct] % The harvested DICOM information

% The geometry set G contains:
% G =

%                                 v: [3x1 double] %The voxel size
%                                 OR: [3x1 double] %The location of the origin
%                                 r: [3x1 double] %Direction vector for rows
%                                 c: [3x1 double] %Direction vector for Columns

pathName='C:\Users\simulator II\Documents\experimentalData\TrioTim-35469-20170706-094232-015000';

%Get all subfolders
if ispc
    pathNames = regexp(genpath(pathName),[filesep,';'], 'split');
elseif isunix
    pathNames = regexp(genpath(pathName),':', 'split');
else

125
pathNames = regexp(genpath(pathName), [filesep, ';'], 'split');
end

pathNames = pathNames(1:end-1)';
numberOfFolders = numel(pathNames);

% Converting DICOM data to IMDAT format in all subfolders
for q = 1:1:numberOfFolders
    pathNameSub = pathNames{q}; % Current path name
    if isempty(strfind(pathNameSub, [filesep, 'IMDAT'])); % if the IMDAT directory does not exist yet
        try
            dcmFolder2MATobject(pathNameSub, []); % Get DICOM data
        catch exception
            warning([exception.message, '-> Analysis skipped for: ', pathName]);
        end
    end
end

% Example: LOADING OR HANDLING THE MAT OBJECT
% Here is an example for loading in the entire data structure

loadName = fullfile(pathName, '8', 'IMDAT', 'IMDAT.mat');
IMDAT_struct = load(loadName);

%Indexing into the MAT object to avoid loading entire structure
% In some cases it is not desirable to load in the entire data set but only
% say a certain slice. In this case the MAT object allows for indexing as
% shows below. See also the help documentation for |matfile|
% Although this type of indexing can be slow it does allow one to only select a subset of the data which in some cases helps to save memory
matObj = matfile(loadName);
G = matObj.G;
M = matObj.type_1;

%%
% Viewing the image data
sv2(M);

%%
% Viewing the image data using |ind2patch|
% Alternatively the image data can be viewed using the |ind2patch|
% function. See the associated help for more information.

%%
%
% <gibbVerySmall.gif>
%
% _*_GIBBON_*_
% <www.gibboncode.org>
%
% _Kevin Mattheus Moerman_, <gibbon.toolbox@gmail.com>
A.3.2 View Converted DICOM Data

% This file uses Gibbon for plotting, https://www.gibboncode.org/
% Gibbon license: GNU General Public License v3.0
% License: MIT License, No commercial use
% author: Kevin Moerman, kmoerman@mit.edu

pathName='C:\Users\simulator II\Documents\experimentalData\TrioTim-35469-20170706-094232-015000';
loadName=fullfile(pathName,'4', 'IMDAT', 'IMDAT.mat');
IMDAT_struct=load(loadName);

matObj = matfile(loadName);
G = matObj.G;
M= matObj.type_1;

% Viewing the image data

sliceNum=5;

logicPlot=false(size(M));
logicPlot(:,:,sliceNum)=1;
[F,V,C]=im2patch(M,logicPlot, 'sk');
[timt
[V(:,1),V(:,2),V(:,3)]=im2cart(V(:,2),V(:,1),V(:,3),G.v);

clf;
hold on;
gpatch(F,V,C,'none',0.8);
axisGeom; view(2);
colormap(gray(250));
drawnow;

%%
P=zeros(2,3);
for q=1:1:2
    [x,y]=ginput(1);
    p=[x,y,mean(V(:,3))];
    P(q,:)=p;
    plotV(p,'r.','MarkerSize',25);
end

%%
d=sqrt(sum((P(1,:)-P(2,:)).^2))
A.4 DIC

A.4.1 Compare Results, in Phantom

% This file uses Gibbon for plotting, https://www.gibboncode.org/,
Gibbon license: GNU General Public License v3.0
% License: MIT License, No commercial use
clear
%% Constants
fontSize = 16;
font.weight = 'b';
line.width = 2;

%% Camera time is obtained from the images
cameratime1 = 3:5:38;
cameratime2 = 1:5:36;
cameraTimes = [cameratime1; cameratime1; cameratime1; cameratime2];

%% Load data
spotnames = {'26', '46', '46t2', '76'};
for i = 1:length(spotnames)
    for i = 1:length(spotnames)
        spotDepth = str2double(spotnames{i}(1:2));
        DICdata = load(strcat('DIC/step', spotnames{i}, 'result2.mat'));
        DICdata = DICdata.zValHighestPoint;
        Pdata = load(strcat('probe/step', spotnames{i}, 'result.mat'));
        M = Pdata.M;
        time = M(:,1);
        depth = M(:,2);

        figure;
        yyaxis left
        plot(cameratimes(i,:), DICdata - DICdata(2), 'o-', 'LineWidth',
            line.width);
ylabel(strcat('step', spotnames{i}, 'DIC data'));
set(gca, 'fontsize', font_size, 'fontweight', font_weight);
ylim([-10 0])

disp(DICdata - DICdata(2));

yyaxis right
plot(time, depth-spot_depth, 'LineWidth', line.width);
ylabel(strcat('step', spotnames{i}, 'My data'));
set(gca, 'fontsize', font_size, 'fontweight', font_weight);
grid on
grid minor
ylim([-10 0])

xlabel('Time (sec)');
title(strcat('My Result vs. DIC Result in step', spotnames(i)));
end

%% Details in CompareResults.xlsx
error = [-0.198
-0.1429
-0.1996
-0.5673
-0.476
-1.6781
-1.0816
-2.3483
-3.3524
1.83
2.2351
2.2492
1.2663
0.2545
0.1931
0.138
-0.1121

131
-0.183
-0.1215
-0.2837;

figure;
ploterrhist(error,'bins',30);
xlim([-4 4])
title('Error Histogram of Indentation Experiment in Phantom using DIC result as ground truth','FontSize',16)
set(gca,'fontsize',24,'fontweight','b');

disp(mean(error));
disp(std(error));
A.4.2 Compare Results, \textit{in-Vivo}

% This file uses Gibbon for plotting, https://www.gibboncode.org/,
% Gibbon license: GNU General Public License v3.0
% License: MIT License, No commercial use

clc

%%% Constants
font.size = 16;
font.weight = 'b';
line.width = 2;

%%% Camera time is obtained from the images
cameratime1 = 7:5:57;
cameratime2 = 3:5:68;
cameratime3 = 5:5:65;
cameratime4 = 4:5:54;
cameratime5 = 4:5:54;
cameraTimes = {cameratime1, cameratime2, cameratime3, cameratime4, cameratime5};

%%% load data
spotnames = {'1','2','3','4','5'};
for i = 1:length(spotnames)
    spot.depth = str2double(spotnames{i});

    DICdata = load(strcat('DIC/trial',spotnames{i},'result.mat'));
    DICdata = DICdata.indentations;
    disp(DICdata)
    Pdata = load(strcat('probe/trial',spotnames{i},'result.mat'));
    M = Pdata.M;
    time = M(:,1);
    depth = M(:,2);
```matlab
figure;
yyaxis left
plot(cameraTimes{i}, DICdata, 'o-', 'LineWidth', line_width);
ylabel(strcat('step', spotnames{i}, ' DIC data'));
set(gca, 'fontsize', font_size, 'fontweight', font_weight);
ylim([-10 5])

yyaxis right
plot(time, depth-spot-depth, '+', 'LineWidth', line_width);
ylabel(strcat('step', spotnames{i}, 'My data'));
set(gca, 'fontsize', font_size, 'fontweight', font_weight);
grid on
grid minor
ylim([85 100])

dateplot('Time (sec)');
title(strcat('My Result vs. DIC Result in trial', spotnames{i}));

waitforbuttonpress
end

%% Details in CompareResults.xlsx
error = [-0.7502
0.6394
0.1895
0.594
0.3353
-0.4738
0.4143
1.3576
0.4175
-0.0632
0.1403
0.4944
0.7322
0.6138
```
-0.4811
-0.3615
-0.0833
-0.8664
-0.1971
0.6463
0.5013
0.2443
0.0107
0.4228
-0.1034
-0.0659
0.1041
-0.475
-0.1707
0.1549

figure;
ploterrhist(error,'bins',30);
xlim([-2 2])
title('Error Histogram of Indentation Experiment in-vivo using DIC result as ground truth','FontSize',16)
set(gca,'fontsize', 24,'fontweight','b');

disp(mean(error));
disp(max(abs(error)));
disp(min(abs(error)));
disp(std(error));
A.4.3 Find Indentation

% License: MIT License, No commercial use

% L = 16.7314/1.102;\% cm distance from middle of the highest two points
% selected to the probe head, this is estimated in Photoshop from
% image

%% Load RecP3D files
for trialnamenum = 1:5
    trailname = strcat('trial',num2str(trialnamenum));
    data = load(strcat(trailname,'/probeRecP3D.mat'));
    data = data.RecP3D;

    Ncams=2;
    nCamR=1;
    nCamL=2;
    Np=6; \% number of points for centroids

    currentPath = pwd;
    imagePath = [currentPath strcat('/',trailname)];

    rightImageSet = imageSet([imagePath '/' num2str(nCamR)]);
    leftImageSet = imageSet([imagePath '/' num2str(nCamL)]);

    rightImageFileNames = rightImageSet.ImageLocation;
    leftImageFileNames = leftImageSet.ImageLocation;

    NumImRight=numel(rightImageFileNames);
    NumImLeft=numel(leftImageFileNames);
    if NumImLeft~=NumImRight, warning('number of images in each camera
        should be equal'); end
indentations = zeros(NumImRight,1);
probehead = zeros(NumImRight,3);
distances = zeros(NumImRight-1,1);
for ii = 1:NumImRight
    mid.high = (data{ii}(1,:)+data{ii}(2,:))/2;
    mid.low = (data{ii}(3,:)+data{ii}(4,:))/2;
    vec = mid.low - mid.high;
    vec = vec/norm(vec);
    if ii == 1
        vec1 = vec;
    end
    probe_est = mid.high + vec*L; % Get the estimate (xyz) of the
probes head
    if ii == 1
        this.indentation = 0;
    else
        this.indentation = dot(probe_est - probehead(ii,:),10, vec1);
    end
    indentations(ii) = this.indentation*10; % cm to mm
    probehead(ii,:) = probe_est*10; % cm to mm
end

%% Plot
fig = figure;
plot(indentations,'o-');
grid on
grid minor
title(strcat(trailname,' indentation'));
set(gca,'fontsize',16,'fontweight','b');
%% Save
saveas(fig, strcat(trailname, 'result.fig'))

%%
save([imagePath 'result.mat'], 'indentations');

end
A.4.4 Process Raw Images

The author of all code in this sub-section is Dana Solav, danask@mit.edu.

Reconstruction

```matlab
function P3D = DLTllReconstruction(P1,P2,L1,L2)

% License: MIT License, No commercial use
% author: Dana Solav, danask@mit.edu

% this function returns the reconstructed 3D coordinates of points as
% measured by 2 cameras P1 and P2, with the respective DLT
% 11 parameters vectors L1 and L2

dim=size(P1,1);
for ii=1:dim % loop over number of mutual points

M = [P1(ii,1)*L1(9)-L1(1) P1(ii,1)*L1(10)-L1(2) P1(ii,1)*L1(11)-L1(3);
P1(ii,2)*L1(9)-L1(5) P1(ii,2)*L1(10)-L1(6) P1(ii,2)*L1(11)-L1(7);
P2(ii,1)*L2(9)-L2(1) P2(ii,1)*L2(10)-L2(2) P2(ii,1)*L2(11)-L2(3);
P2(ii,2)*L2(9)-L2(5) P2(ii,2)*L2(10)-L2(6) P2(ii,2)*L2(11)-L2(7)];

V = [L1(4)-P1(ii,1)
     L1(8)-P1(ii,2)
     L2(4)-P2(ii,1)
     L2(8)-P2(ii,2)];

P3D(ii,:) = M/V;
end
```
end
Step 1 Generate Cylinder Coordinates

% This file uses Gibbon for plotting, https://www.gibboncode.org/, Gibbon license: GNU General Public License v3.0
% License: MIT License, No commercial use
% author: Dana Solav, danask@mit.edu

%%
% Generate 3D cylinder coordinates for the entire cylinder as well as for
% each camera and save into CylinderCoor.mat and CylinderCoorCam.mat

clear all; close all;

% Path where to save the cylinder coordinates
savePath=pwd;
saveTxt=0;

% Calculate calibration point coordinates
Ncam=2; % Number of cameras
firstColAll=[13 12];

nColCam=7; % number of columns seen in each camera
overlap=6; % overlap of column between images
Nr=10; % number of rows
Nc=23; % number of columns
R=7.3/2; % radius of cylinder
dx=1; % distance between columns
dy=1.4; % distance between rows
theta=dx/R; % angle between columns
t=0:theta:2*pi;
t=t(1:Nc);
z=fliplr(0:dy:(Nr-1)*dy);
x=R*cos(t);
y=R*sin(t);
CylinderCoor = zeros(3, Nr, Nc);
for icol = 1:Nc
    for irow = 1:Nr
        CylinderCoor(:, irow, icol) = [x(icol) y(icol) z(irow)];
    end
end

% Convert into array and plot
CylinderCoorArray = reshape(CylinderCoor, 3, Nc*Nr);

% plot
cFigure
plotV(CylinderCoorArray, '.k');
numP = arrayfun(@num2str, 1:Nc*Nr, 'unif', 0);
text(CylinderCoorArray(:, 1), CylinderCoorArray(:, 2), CylinderCoorArray(:, 3), numP, 'color', 'r');
axisGeom

% Write coordinates for each camera based on first and last columns visible
colCaminitial = 1:Nc;
for icam = 1:Ncam
    colCamtemp = circshift(colCaminitial, Nc-firstCol+(icam-1)*(nColCam-overlap)+1);
    colCamtemp = circshift(colCaminitial, Nc-firstColAll(icam)+1);
    colCam(icam, :) = colCamtemp(1:nColCam);
end
CylinderCoorCam = cell(Ncam, 1);
for icam = 1:Ncam
    CylinderCoorCam{icam} = CylinderCoor(:, :, colCam(icam,:));
end

% Write coordinates for each pair of cameras based on first and last columns visible
for ipair = 1:Ncam
% colCamtemp = circshift(colCaminitial, Nc-firstCol+(ipair-1)*(nColCam -overlap)+1);
colCamtemp = circshift(colCaminitial, Nc-firstColAll(ipair)+1);
colCamPair(ipair,:) = colCamtemp(1:overlap);

end
CylinderCoorCamPairs=cell(Ncam,1);
for ipair=1:Ncam
    CylinderCoorCamPairs{ipair}=CylinderCoor(:,:,colCamPair(ipair,:));
end

save([savePath '\CylinderCoor.mat'],'CylinderCoor');
save([savePath '\colCam.mat'],'colCam');
save([savePath '\CylinderCoorCam.mat'],'CylinderCoorCam');
save([savePath '\colCamPair.mat'],'colCamPair');
save([savePath '\CylinderCoorCamPairs.mat'],'CylinderCoorCamPairs');

if saveTxt==1
    % write text files
    csvwrite('colCam.txt',colCam);
csvwrite('colCamPair.txt',colCamPair);
csvwrite('CylinderCoorAll.txt',CylinderCoorArray);
for icam=1:Ncam
    CylinderCoorCamNow=CylinderCoorCam{icam};
    CylinderCoorCamArray=reshape(CylinderCoorCamNow,3,size(CylinderCoorCamNow,2)*size(CylinderCoorCamNow,3))';
csvwrite([savePath '\CylinderCoorCam' num2str(icam) '.txt'],
           CylinderCoorCamArray);
end
    camCyc=[1:Ncam 1];
for ipair=1:Ncam
    CylinderCoorPairNow=CylinderCoorCamPairs{ipair};
    CylinderCoorPairArray=reshape(CylinderCoorPairNow,3,size(CylinderCoorPairNow,2)*size(CylinderCoorPairNow,3))';
csvwrite([savePath '\CylinderCoorPair' num2str(camCyc(ipair)) '-'
              ' num2str(camCyc(ipair+1)) '.txt'],CylinderCoorCamPairArray);
Step 2 Undistort and Find Centroids

% This file uses Gibbon for plotting, https://www.gibboncode.org/,
Gibbon license: GNU General Public License v3.0
% License: MIT License, No commercial use
% author: Dana Solav, danask@mit.edu

%%
% for each camera, undistort the calibration image based on the
parameters found with FindDistortionPar
% then crop it and turn it into BW
% Then, find the 2D centroids on each image and save to
Centroids.unsorted.mat
% Then, Sort the 2D positions of the centroids according to number of
row and columns and save to 'Centroids.sorted.mat'
% Then, calculate the DLT calibration parameters based on the known 3D
positions of the points

clear all; close all

% save centroids, images, and DLT parameters?
Ncam=2;
indCams=[1 2];%Ncam-1
saveOn=1; %save camera parameters?

currentPath = pwd;
% Path where images are located. Inside this folder, the images should
% be ordered IM1...IMn
imagePath=[currentPath 'Cylinder-images'];
% Path where to save the camera parameters
savePath=currentPath;

Nc=7; % number of columns
Nr=10; % number of rows
Np=Nr*Nc; %number of points in full calibration image
%% load undistorted images, undistort and crop them, and find centroids
load('colCam.mat');
load('paramsAllcams.mat');
camParams{1}=paramsAllcams{3};
camParams{2}=paramsAllcams{4};

for icam=indCams

    colFirst=colCam(icam,1);
colLast=colCam(icam,end);

    IM = imread(['imagePath ' num2str(icam) '.jpg']);

    % turn to gray
    IMgr = rgb2gray(IM);

    r=size(IMgr,1); %rows
c=siz3(IMgr,2); %columns

    % undistort image based on camera parameters obtained by
    % checkerboard calibration
    %params=paramsAllcams{icam};
camParams=camParams{icam}; % we currently have only parameters for 1
camera
    [IMgrUD,newOrigin] = undistortImage(IMgr,params);

    if saveOn==1
        imwrite(IMgrUD, ['imagePath \IMgrUD.cam' num2str(icam) '.jpg']);
    end

    %plot original and undistorted
    fh=figure;
    fh.Units='normalized'; fh.Position=[.02 .1 .9 .8];
    subplot(1,2,1)
    imshow(IMgr); zoom(1.5)
    imshow(IMgrUD); zoom(1.5)
title('Original Image');
subplot(1,2,2)
imshow(IMgrUD); zoom(1.5)
title('Undistorted Image');
suptitle(['Camera ' num2str(icam)]);
waitforbuttonpress

% increase image resolution for better centroid finding
resFactor=3;
IMgrUDhr = imresize(IMgrUD,resFactor);

centroidsOK=0; % check if centroids are correct
while ~centroidsOK
    % crop polygon and turn to white everything out of polygon
    polyOK=0;
    while ~polyOK
        fh=figure;
        fh.Units='normalized'; fh.Position=[.02 .1 .9 .8];
        title([' draw polygon around columns ' num2str(colFirst) ' to ' num2str(colLast)]); hold on
        [BW,xi,yi] = roipoly(IMgrUDhr);
        polyOK=input('polygon OK?');
    end

    % turn to white everything outside the polygon
    IMgrcr=IMgrUDhr;
    IMgrcr(~BW)=255;

    % convert to BW
    level = multithresh(IMgrcr,2);
    IMbw = imbinarize(uint8(IMgrcr),double(level(1))/255);

    % plot images
    fh=figure;
    fh.Units='normalized'; fh.Position=[.02 .1 .9 .8];
    subplot(1,3,1)
imshow(IMgrUD); zoom(1.5)
title('Grayscale undistorted Image');
subplot(1,3,2)
imshow(IMgrcr); zoom(1.5)
title('Cropped Image');
subplot(1,3,3)
imshow(IMbw); zoom(2)
title('BW Image');

waitforbuttonpress

levelOK=0; % check if centroids are correct
while ~levelOK

IMbw = imbinarize(uint8(IMgrcr),double(level(1))/255);

% remove salt and pepper noise
J = medfilt2(IMbw,[3 3]);
% remove black pixels at the image corners
J(1,1)=0; J(1,c)=0; J(r,c)=0; J(r,1)=0;
% transform 0 in 1 and viceversa
J=-J;

% label the connected regions in the image (each "dot" is a region labeled with a different number in L, and num is the number of regions)
[L,num] = bwlabeled(J);

% find the geometrical properties (in our case the centroids and areas) of regions in the image
Centroids = regionprops(L,'Centroid'); % a struct of num fields, each one is a 2D centroid position
Areas = regionprops(L,'Area');
for k=1:numel(Centroids)
  C(k,:)=Centroids(k).Centroid;
end
AreasArray = struct2array(Areas);
[Asort, indA] = sort(AreasArray);

% delete the centroids with the smallest area
if num -= Np
    warning('wrong number of points');
    C(indA(1:num-Np), :) = [];
end

% plot BW images over original with centroids on figures
fh = figure; hold all
fh.Units = 'normalized'; fh.Position = [.02 .1 .9 .8];
ax1 = axes;
hg = imagesc(IMgrUDhr); hold on
ax2 = axes;
hb = imagesc(J); hold on
linkaxes([ax1, ax2]); % Link them together
% Hide the top and bottom axes
ax1.Visible = 'off'; pbaspect(ax1, [size(IMgrUD, 2) size(IMgrUD, 1)])
ax1.XTick = []; ax1.YTick = [];
ax2.Visible = 'off'; pbaspect(ax2, [size(IMgrUD, 2) size(IMgrUD, 1)])
ax2.XTick = []; ax2.YTick = [];
colormap(ax1, 'gray'); colormap(ax2, 'parula'); % Give each one its own colormap
% set transparency
alphaValue = 0.5;
alpha.data = double(J);
alpha.data(alpha.data ~= 0) = alphaValue;
set(hb, 'AlphaData', alpha.data);
set(ax2, 'Position', ax1.Position);
% plot centroids
plot(C(:, 1), C(:, 2), 'r+', 'LineWidth', 1, 'MarkerSize', 5);
zoom(2)
waitforbuttonpress

149
levelOK=input(['level is ' num2str(level(1)) '. level OK?
press 1 if yes, 0 if no ']);
if ~levelOK
    level(1)=input('enter new level');
end

centroidsOK=input('centroids OK?');
end
if saveOn==1
    imwrite(IMbw,[imagePath '\IMBW.cam' num2str(icam) '.jpg']);
end

% save centroids in cell
Centroids.unsorted{icam}=C/resFactor;

close all
clear C

end
if saveOn==1
    save([savePath '\Centroids.unsorted.mat'], 'Centroids.unsorted');
end

%% Sort the 2D positions of centroids on a calibration image

Centroids.sorted=cell(Ncam,1);
load([savePath '\Centroids.unsorted.mat']);

for icam=indCams:Ncam
    % load unsorted centroid list file and plot on image
    P = Centroids.unsorted{icam};
    I0 = imread([imagePath '\IMgrUD.cam' num2str(icam) '.jpg']);

    fh=figure;
imshow(IO); zoom(1.5); hold on
title([Camera ' num2str(icam)]);

% sort the columns by the distance between each point and the line
% between the corners
p3=P';
p3(3,:)=0; % P in 3D format
NpNew=Np; % New number of points (1 column is deleted every loop)
PS=zeros(size(P)); % matrix for sorted points
for icol=1:Nc
    % Upper left and lower left coordinates
    [Xmin,XminInd]=min(p3(1,:));
    [Xmax,XmaxInd]=max(p3(1,:));
    [Ymin,YminInd]=min(p3(2,:));
    [Ymax,YmaxInd]=max(p3(2,:));

    if icam==1
        c1=p3(1:2,YminInd);
        c2=p3(1:2,XminInd);
    else
        c1=[Xmin; Ymin];
        c2=[Xmin; Ymax];
    end
    % plot these points
    plotV([c1,c2]', '+c');

    % find the 2 points with minimum distance to top left corner
    % and choose the one with the smaller X
    Dist1 = pdist2(c1',p3(1:2,:'));
    [~,Dist1SortInd]=sort(Dist1);
    Min2Dist1SortInd=Dist1SortInd(1:2);
    [~,MinXSortInd]=min([p3(1,Min2Dist1SortInd(1)) p3(1, Min2Dist1SortInd(2))]);
minIND1=Min2Dist1SortInd(MinXSortInd);

% find the 2 points with minimum distance to bottom left corner and choose the one with the smaller X
Dist2 = pdist2(c2', p3(1:2,:)) ;
[~,Dist2SortInd]=sort(Dist2);
Min2Dist2SortInd=Dist2SortInd(1:2);
[~,MinXSortInd]=min([p3(1,Min2Dist2SortInd(1)) p3(1,Min2Dist2SortInd(2))]);
minIND2=Min2Dist2SortInd(MinXSortInd);

pl=p3(:,minIND1); % first point of the line
p2=p3(:,minIND2); % second point of the line
plot([p1(1);p2(1)],[p1(2);p2(2)],'.m')

d=[]; % distance between all remaining points and the straight line
for ip=1:NpNew
    d(ip)= norm(cross(pl-p2,p3(:,ip)-p2))/norm(pl-p2); % distance between each point in P and the line
end
[dSorted,dSortedInd]=sort(d); % sort distance to line
PStemp=p3(1:2,dSortedInd(1:Nr))'; % take the Nr points closest to the line- these are the points in the current column
PStemp=sortrows(PStemp,2); % sort these points by Y
PS((Nr*(icol-1)+1):Nr*icol,:)=PStemp; % save this column into PS
p3(:,dSortedInd(1:Nr))=[]; % delete these points from p3
NpNew=NpNew-Nr; % update number of remaining points
end

plot(PS(:,1),PS(:,2),'+r','markersize',3)
waitforbuttonpress
plot(PS(:,1),PS(:,2),'b','linewidth',1);
waitforbuttonpress

152
numCams = arrayfun(@num2str, 1:Np, 'unif', 0);
text(PS(:,1), PS(:,2), numCams, 'color', 'g', 'fontsize', 10);
waitforbuttonpress

close all;

% save centroids in cell
Centroids_sorted{icam} = PS;

end

if saveOn == 1
    save([savePath 'Centroids_sorted.mat'], 'Centroids_sorted');
end

%% Direct linear transformation from the 2D image to 3D target points ( 
save DLT transformation matrix L for each camera)

DLTpar = cell(Ncam, 1);
load([savePath 'Centroids_sorted.mat']);
load([savePath 'CylinderCoorCam.mat']);

for icam = indCams:Ncam
    % 2D coordinates from image
    C2D = Centroids_sorted{icam};

    % 3D true points coordinates
    C3Dtemp = CylinderCoorCam{icam};
    C3D = reshape(C3Dtemp, 3, Np);'

    % Solve the linear system with the LS method for the unknown 11 DLT parameters that reflect the relationships between the world 3D and the image 2D (vector L)
    % arrange the coordinates in R in one array
    L = DLT11Calibration(C2D, C3D);

    % save L with cam number
DLTpar{icam}=L;

end

if saveOn==1
    save([savePath 'DLTpar.mat'], 'DLTpar');
end
Step 4 Find Markers and Reconstruct

% This file uses Gibbon for plotting, https://www.gibboncode.org/,
% Gibbon license: GNU General Public License v3.0
% License: MIT License, No commercial use
% author: Dana Solav, danask@mit.edu

%%% calculate 3D positions of point clouds based on DLT parameters
clear all
close all
clc

Ncams=2;
nCamR=1;
nCamL=2;
Np=6; % number of points for centroids

%%
currentPath = pwd;
imapPath = [currentPath '/trial1'];

%% Load target images
% Specify calibration images
rightImageSet = imageSet([imagePath ' /' num2str(nCamR)]);
leftImageSet = imageSet([imagePath ' /' num2str(nCamL)]);

rightImageFileNames = rightImageSet.ImageLocation;
leftImageFileNames = leftImageSet.ImageLocation;

NumImRight=numel(rightImageFileNames);
NumImLeft=numel(leftImageFileNames);
if NumImLeft~=NumImRight, warning('number of images in each camera should be equal'); end
% Nsubplots=numSubplots(NumImLeft);
%% Rotate images

% mkdir([imagePath '/' num2str(nCamR) '/rotated']);

Iright=cell(NumImRight,1);
for ii=1:NumImRight
    Iright{ii}=imread(rightImageFileNames{ii});
    Iright{ii}=imrotate(Iright{ii},-90);
    imwrite(Iright{ii}, [imagePath '/' num2str(nCamR) '/rotated/I'
        num2str(ii,'%02i') '.jpg']);
end

% mkdir([imagePath '/' num2str(nCamL) '/rotated']);

Ileft=cell(NumImLeft,1);
for ii=1:NumImLeft
    Ileft{ii}=imread(leftImageFileNames{ii});
    Ileft{ii}=imrotate(Ileft{ii},-90);
    imwrite(Ileft{ii}, [imagePath '/' num2str(nCamL) '/rotated/I'
        num2str(ii,'%02i') '.jpg']);
end

% leftImageSet = imageSet([imagePath '/' num2str(nCamR) '/rotated']);
% rightImageSet = imageSet([imagePath '/' num2str(nCamL) '/rotated']);

% leftImageFileNames = leftImageSet.ImageLocation;
% rightImageFileNames = rightImageSet.ImageLocation;

%% undistort images

% load('camParams4.mat'); load('camParams3.mat');
load('paramsAllcans.mat');
camParams3 = paramsAllcans(3);  
camParams4 = paramsAllcans(4);  
camParams3 = camParams3{l};    
camParams4 = camParams4{l};

% Right

camParams=camParams3;
Jright=cell(NumImRight,1);
newOriginRight = cell(NumImRight, 1);
mkdir([imagePath '/' num2str(nCamR) ' undistorted']);
for ii=1:NumImRight
    [Jright{ii},newOriginRight{ii}] = undistortImage(Iright{ii}, camParams,'OutputView','same');
imwrite(Jright{ii}, [imagePath '/' num2str(nCamR) ' undistorted/J' num2str(ii, '%02i') '.jpg']);
end

% Left
.camParams=camParams4;
Jleft = cell(NumImLeft, 1);
newOriginLeft = cell(NumImLeft, 1);
mkdir([imagePath '/' num2str(nCamL) ' undistorted']);
for ii=1:NumImLeft
    [Jleft{ii},newOriginLeft{ii}] = undistortImage(Ileft{ii}, camParams,'OutputView','same');
imwrite(Jleft{ii}, [imagePath '/' num2str(nCamL) ' undistorted/J' num2str(ii, '%02i') '.jpg']);
end

%% load undistorted images and crop to find centroids
mkdir([imagePath '/' num2str(nCamR) ' bw']);
mkdir([imagePath '/' num2str(nCamL) ' bw']);
Centroids.unsorted.R = cell(NumImLeft, 1);
Centroids.unsorted.L = cell(NumImLeft, 1);
for ii=1:NumImLeft
    % turn to gray
    IMRgr = rgb2gray(Jright{ii});
    IMLgr = rgb2gray(Jleft{ii});
    r=size(IMRgr, 1); %rows
    c=size(IMRgr, 2); %columns

157
% increase image resolution for better centroid finding
resFactor=3;
IMRgrHD = imresize(IMRgr,resFactor);
IMLgrHD = imresize(IMLgr,resFactor);

% CENTROID RIGHT
centroidsOK=0; % check if centroids are correct
while ~centroidsOK
    % crop polygon and turn to white everything out of polygon
    polyOK=0;
    while ~polyOK
        fh=figure; fh.Units='normalized'; fh.Position=[.02 .1 .9 .8];
        title([ 'Draw polygon around ROI of Right image ' num2str(ii) ]);
        hold on
        [BW,xi,yi] = roipoly(IMRgrHD);
        polyOK=input('polygon OK?');
    end

    % turn to white everything outside the polygon
    IMRcrop=IMRgrHD;
    IMRcrop(~BW)=255;

    % convert to BW
    level = multithresh(IMRcrop,2);
    IMRbw = imbinarize(uint8(IMRcrop),double(level(1))/255);

    % plot images
    fh=figure;
    fh.Units='normalized'; fh.Position=[.02 .1 .9 .8];
    subplot(1,3,1)
    imshow(IMRgrHD); zoom(1.5)
    title('Grayscale undistorted Image');
    subplot(1,3,2)
    imshow(IMRcrop); zoom(1.5)
    title('Cropped Image');
```matlab
subplot(1,3,3)
imshow(IMRbw); zoom(2)
title('BW Image');
suptitle(['Right image ' num2str(ii)]);
waitforbuttonpress

levelOK=0; % check if centroids are correct
while ~levelOK

    IMRbw = imbinarize(uint8(IMRcrop),double(level(l))/255);

    % remove salt and pepper noise
    J = medfilt2(IMRbw,[3 3]);

    % remove black pixels at the image corners
    J(1,1)=1; J(1,c)=1; J(r,c)=1; J(r,1)=1;

    % trasform 0 in 1 and viceversa
    J=-J;

    % label the connected regions in the image (each "dot" is a region labeled with a different number in L, and num is the number of regions)
    [L,num] = bwlabel(J);

    % find the geometrical properties (in our case the centroids and areas) of regions in the image
    Centroids = regionprops(L,'Centroid'); % a struct of num fields, each one is a 2D centroid position
    Areas = regionprops(L,'Area');
    for k=1:numel(Centroids)
        C(k,:)=Centroids(k).Centroid;
    end
    AreasArray=struct2array(Areas);
    [Asort,indA]=sort(AreasArray);

    % delete the centroids with the smalllets area
```
if num==Np
    warning('wrong number of points');
    C(indA(1:num-Np),:)=[];
end

% plot BW images over original with centroids on figures
fh=figure; hold all
fh.Units='normalized'; fh.Position=[.02 .1 .9 .8];
ax1 = axes;
hg=imagesc(IMRgrHD); hold on
ax2 = axes;
hb=imagesc(J); hold on
linkaxes([ax1,ax2]); % Link them together
% Hide the top and bottom axes
ax1.Visible = 'off'; pbaspect(ax1,[size(IMRgr,2) size(IMRgr,1) 1])
ax1.XTick = []; ax1.YTick = [];
ax2.Visible = 'off'; pbaspect(ax2,[size(IMRgr,2) size(IMRgr,1) 1])
ax2.XTick = []; ax2.YTick = [];
colormap(ax1,'gray'); colormap(ax2,'parula'); % Give each
    one its own colormap
%set transparancy
alphaValue=0.5;
alpha.data=double(J);
alpha.data(alpha.data==0)=alphaValue;
set(hb, 'AlphaData', alpha.data);
set(ax2,'Position',ax1.Position);
% plot centroids
plot(C(:,1),C(:,2),'r+','LineWidth',1,'MarkerSize',5);
zoom(2)
waitforbuttonpress

levelOK=input(['level is ' num2str(level(l)) '. level OK?
    press 1 if yes, 0 if no ']);
if ~levelOK
level(1)=input('enter new level');
end

centroidsOK=input('centroids OK?');
end

% save bw image
imwrite(IMRbw,[imagePath '/num2str(nCamR) '/bw/K' num2str(ii,'%02i') '.jpg']);

% save centroids in cell
Centroids.unsorted_R{ii}=C/resFactor;
clear C

% CENTROID LEFT
centroidsOK=0; % check if centroids are correct
while ~centroidsOK
    % crop polygon and turn to white everything out of polygon
    polyOK=0;
    while ~polyOK
        fh=figure; fh.Units='normalized'; fh.Position=[.02 .1 .9 .8];
        title(['Draw polygon around ROI of Left image ' num2str(ii)]);
        hold on
        [BW,xi,yi] = roipoly(IMLgrHD);
        polyOK=input('polygon OK?');
    end

    % turn to white everything outside the polygon
    IMLcrop=IMLgrHD;
    IMLcrop(~BW)=255;

    % convert to BW
    level = multithresh(IMLcrop,2);
    IMLbw = imbinarize(uint8(IMLcrop),double(level(1))/255);

    % plot images

161
fh=figure;
fh.Units='normalized'; fh.Position=[.02 .1 .9 .8];

subplot(1,3,1)
imshow(IMLgrHD); zoom(1.5)
title('Grayscale undistorted Image');

subplot(1,3,2)
imshow(IMLcrop); zoom(1.5)
title('Cropped Image');

subplot(1,3,3)
imshow(IMLbw); zoom(2)
title('BW Image');
suptitle(['Left image ' num2str(ii)]);

waitforbuttonpress

levelOK=0; % check if centroids are correct
while ~levelOK

    IMLbw = imbinarize(uint8(IMLcrop),double(level(1))/255);

    % remove salt and pepper noise
    J = medfilt2(IMLbw,[3 3]);
    % remove black pixels at the image corners
    J(1,1)=1; J(1,c)=1; J(r,c)=1; J(r,1)=1;
    % transform 0 in 1 and viceversa
    J=~J;

    % label the connected regions in the image (each "dot" is a region labeled with a different number in L, and num is the number of regions)
    [L,num] = bwlabel(J);

    % find the geometrical properties (in our case the centroids and areas) of regions in the image
    Centroids = regionprops(L,'Centroid'); % a struct of num fields, each one is a 2D centroid position

end
Areas = regionprops(L,'Area');
for k=1:numel(Centroids)
    C(k,:) = Centroids(k).Centroid;
end
AreasArray = struct2array(Areas);
[Asort, indA] = sort(AreasArray);

% delete the centroids with the smallest area
if num==Np
    warning('wrong number of points');
    C(indA(1:num-Np),:) = [];
end

% plot BW images over original with centroids on figures
fh = figure; hold all
fh.Units='normalized'; fh.Position=[.02 .1 .9 .8];
axl = axes;
hg = imagesc(IMLgrHD); hold on
ax2 = axes;
hb = imagesc(J); hold on
linkaxes([axl, ax2]); % Link them together
% Hide the top and bottom axes
axl.Visible = 'off'; pbaspect(axl,[size(IMLgr,2) size(IMLgr ,1) 1])
axl.XTick = []; axl.YTick = [];
ax2.Visible = 'off'; pbaspect(ax2,[size(IMLgr,2) size(IMLgr ,1) 1])
ax2.XTick = []; ax2.YTick = [];
colormap(axl,'gray'); colormap(ax2,'parula'); % Give each
    one its own colormap
% set transparency
alphaValue=0.5;
alpha.data = double(J);
alpha.data(alpha.data==0) = alphaValue;
set(hb, 'AlphaData', alpha.data);
set(ax2,'Position',axl.Position);
% plot centroids
plot(C(:,1),C(:,2),'r+','LineWidth',1,'MarkerSize',5);
zoom(2)
waitforbuttonpress

levelOK=input(['level is ' num2str(level(1)) '. level OK?
 press 1 if yes, 0 if no ']);
if ~levelOK
    level(1)=input('enter new level');
end

centroidsOK=input('centroids OK?');
end

% save bw image
imwrite(IMLbw,[imagePath ' /' num2str(nCamL) '/bw/K' num2str(ii,'%02i
') '.jpg']);

% save centroids in cell
Centroids.unsorted_L{ii}=C/resFactor;
clear C
end
close all

% % Sort centroids

Centroids_sorted_R=cell(NumImLeft,1);
Centroids_sorted_L=cell(NumImLeft,1);

for ii=1:NumImLeft

    % load unsorted centroid list file and plot on image
    PR = Centroids.unsorted_R{ii};
    PL = Centroids.unsorted_L{ii};

    % sort the columns by the distance between each point and the
    line between the corners

    .

    164
[-,PRsortInd]=sort(PR(:,2));
PRsorted=PR(PRsortInd,:);
[-,PLsortInd]=sort(PL(:,2));
PLsorted=PL(PLsortInd,:);

% plot
fh=figure;
fh.Units='normalized'; fh.Position=[.02 .1 .9 .8];
subplot(1,2,1)
imshow(Jleft{ii}); hold on; zoom(2)
title(['Left image ' num2str(ii)]);
subplot(1,2,2)
imshow(Jright{ii}); hold on; zoom(2)
title(['Right image ' num2str(ii)]);
Ptext=arrayfun(@num2str,l:Np,'unif',O);
subplot(1,2,1)
plot(PLsorted(:,1),PLsorted(:,2),'+r','markersize',5)
text (PLsorted(:,1),PLsorted(:,2),Ptext,'color','y', fontsize',10);
subplot(1,2,2)
plot(PRsorted(:,1),PRsorted(:,2),'+r','markersize',5)
text(PRsorted(:,1),PRsorted(:,2),Ptext,'color','y','fontsize',10);

% save centroids in cell

Centroids.sorted_R{ii}=PRsorted;
Centroids.sorted_L{ii}=PLsorted;
end

save([currentPath '/Centroids.sorted_R.mat'], 'Centroids.sorted_R');
save([currentPath '/Centroids.sorted_L.mat'], 'Centroids.sorted_L');

% 3D reconstruction using Direct Linear Transformation

% load DLT parameters

165
load([currentPath '/DLTpar.mat']);

% transform 2D points into 3D using DLT

L1=DLTpar{nCamR}; % DLT parameters for 1st camera
L2=DLTpar{nCamL}; % DLT parameters for 2nd camera

for ii=1:NumImLeft
    P1=Centroids.sortedR{ii};
    P2=Centroids.sortedL{ii};

    % Solve the linear system with the LS method for the unknown X Y and Z
    P3D=DLTllReconstruction(P1,P2,L1,L2);

    % Write the reconstructed 3D coordinates to mat file
    RecP3D{ii}=P3D;
end

save([currentPath '/RecP3D.mat'],'RecP3D');

% plot 3D points

cFigure
Colors1=gjet(NumImLeft+1);
for ii=1:NumImLeft
    plotV(RecP3D{ii},'o-','Color',Colors1(ii,:),'markersize',6); hold on
end
axisGeom
Ptext=arrayfun(@num2str,1:NumImLeft,'unif',0);
legend(Ptext);
supertitle('reconstructed points');

%%% p1=RecP3D{1};
%%% p2=RecP3D{2};
%%% p2(:,3)-p1(:,3)
A.5 Other

A.5.1 Plotting support

```matlab
function my_vline(x, color)
    y = get(gca,'ylim');
    color = 'r';
    linewidth = 2;
    plot([x x],y,'Color',color,'LineWidth',linewidth);
end
```

A.5.2 Staircase Calibration

```matlab
% Staircase calibration of speed of sound
true_depth = 16:10:86;
measured_peak_time = [29.8, 50.35, 70.51, 89.68, 111.4, 130.7, 151.2, 172.8];
x = true_depth';
beta = inv(x'*x)*x'*measured_peak_time';
% disp(beta)
linearFitTime = beta*x;
speedOfSound = 1000*86*2/170.7969;

calculated_depth = speedOfSound * measured_peak_time / 2 /1000;

figure;
plot(true_depth, calculated_depth,'kx-',true_depth,true_depth,'k:','LineWidth',1);
legend('calibrated measured depth', 'reference');
xlabel('True Depth [mm]');
ylabel('Measured Depth [mm]');
title('Staircase Phantom Speed of Sound Calibration');
```
grid on
grid minor
set(gca, 'FontSize',16,'FontWeight','b');
Bibliography


