Realization of a Measuring Device for Recording the Relative Movement between Residual Limb and Prosthetic Socket

Aufbau eines Messsystems zur Erfassung der Relativbewegung zwischen Beinstumpf und Prothesesschaft

Master Thesis at the Institute for Mechatronic Systems in Mechanical Engineering
Technische Universität Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

This thesis was presented by

Sigrid Whitmore (2316831)

Supervisor: Veronika Noll
Timeframe: 11/15/2017 until 5/15/2018

Darmstadt, the 15th of May, 2018

Realization of a Measuring Device for Recording the Relative Movement between Residual Limb and Prosthetic Socket

TECHNISCHE UNIVERSITÄT DARMSTADT

IMS

Master's Thesis
Sigrid Whitmore (2316831)
Aufbau eines Messsystems zur Erfassung der Relativbewegung zwischen Beinstumpf und Prothesenschaft

As the connection between human and mechanical system, the residual limb-socket interface has a large degree of influence over the satisfaction and mobility of the user. The production of prosthetic sockets is currently a handicraft. The quality end-product of the iterative process depends measurably upon the expertise of the prosthetist and the subjective feedback of the patient. An objective and quantitative evaluation of the residual limb-socket-interaction does not take place. At the IMS, research is being done to determine if the relative motion between residual limb and socket in dynamic gait situations can objectivize the socket fitting process. To this end, a functional model has been developed and built which enables the experimental measurement of the relative motion in dynamic gait situations.

Within the scope of this master thesis, the functional model, based on the existing concept, shall be further developed into a measuring device. The type of sensor is already defined by the functional model. First, an appropriate microcontroller shall be chosen and programmed. The expansion of the functional model to a measuring device shall take place through the successive construction of a sensor array. During this process, the functionality of the measuring device shall be continuously confirmed. The synchronization of the sensor signals as well as the optimization of data communication with MATLAB has proven to be an especially critical aspect to good function. Depending upon progress, the option to plan and prepare a pilot study with an amputee is available.

Tasks:

- Familiarization with the field
- Choice and programming of an appropriate microcontroller
- Successive construction of the sensor array
- Tests on data transfer and optimization of data communication to MATLAB
- Evaluation of the functionality of the sensor array on a test rig
- Optional: Planning and preparation of the pilot study

Start Date: 15.11.2017

Supervision of the work: Veronika Noll, M.Sc. (IMS)

Darmstadt, 08.11.2017

Prof. Dr.-Ing. S. Rinderknecht

Abstract

Relative motion between residual limb and prosthetic socket is an indication of poor fit. Both the fabrication and fitting processes are highly subjective and a favorable result depends upon the technician's expertise. Although numerous methods exist to measure the relative motion, all have limitations and are not well suited for clinical use. A measurement system using optical sensors has been proposed by students at the Technische Universität Darmstadt and evaluations of a functional model have yielded promising results.

In this thesis, the existing functional model is improved and expanded to use an array of sensors. A new microcontroller is selected and incorporated into the system. The software and data communication are optimized for fast, reliable performance and the system is then evaluated on a test rig to determine favorable calibration settings and quantify performance.

System frequencies up to 1299 Hz are achieved. It is found that the surface microstructure has a dominant effect over short measurement distances; calibrations performed over longer distances are to be preferred. For the chosen calibration factors, the greatest relative errors over a 40 mm distance are found to be 0.90% ± 0.51% in the X direction and –4.76% ± 1.61% in the Y-direction. A systematic drift is also identified.

The final system accommodates up to eight sensors and is controlled from a feature-rich MATLAB GUI.

Kurzfassung

Die Relativbewegung zwischen Beinstumpf und Prothesenschaft ist ein Kennzeichen für eine schlechte Passform. Die Herstellungs- und Anpassungsprozesse sind sehr subjektiv und ein gutes Ergebnis hängt von der Erfahrung des Technikers ab. Obwohl es viele Methoden zur Messung der Relativbewegung gibt, weisen alle Schwächen auf und sind für einen klinischen Einzatz eher ungeeignet. Studenten an der Technischen Universität Darmstadt haben ein Messsystem entwickelt, das auf optische Sensoren basiert und Auswertungen eines Funktionsmodells haben gute Ergebnisse geliefert.

Im Rahmen dieser Arbeit wird das Funktionsmodell verbessert und erweitert um ein Sensorarray benutzen zu können. Ein neuer Mikrokontroller wird ausgewählt und in das System eingebaut. Die Software und Datanübertragung werden optimiert um ein schnelles, zuverlässiges Verhalten zu gewährleisten und das System wird danach an einem Prüfstand untersucht um vorteilhafte Kalibrationseinstellungen zu identifizieren und das Verhalten zu quantifizieren.

Das System erreicht Frequenzen von bis zu 1299 Hz. Die Effekte der Mikrostruktur dominieren bei kleine Messstrecken; es sind für die Kalibration längere Strecken zu benutzen. Für die ausgewählten Kalibrationsfaktoren betragen die größten relative Fehler 0.90% ± 0.51% in X Richtung und –4.76% ± 1.61% in Y Richtung. Ein systematischer Drift wird auch erkannt.

Das System kann mit bis zu acht Sensoren benutzt werden und wird von einer funktionsreichen MATLAB GUI aus gesteuert.

Statements

Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

5/15/2018 — Date

Ingrid Christmann — Signature

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Muster Mustermann, die vorliegende Master-Thesis / Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

15. 5. 2018 — Datum

Ingrid Christmann — Unterschrift

Ich bin damit einverstanden, dass das Urheberrecht an meiner Arbeit zu wissenschaftlichen Zwecken genutzt werden kann.

Darmstadt, 15. Mai 2018 — Ort, Datum

Ingrid Christmann — Unterschrift

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Muster Mustermann, die vorliegende Master-Thesis / Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

English translation for information purposes only:

Thesis Statement pursuant to § 23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Muster Mustermann, have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are identical in content.

For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

Datum / Date: 15.5.2018

Unterschrift/Signature: Muster Mustermann

Table of Contents

Table of Figures

Table of Tables

Table of Codes

Abbreviations

ADP	Advanced Design Project
ARP	Advanced Research Project
GUI	Graphical User Interface
USB	Universal Serial Bus
SPI	Serial Peripheral Interface
SS	Slave Select
MOSI	Master Out/Slave In
MISO	Master In/Slave Out
SCLK	Clock
CPI	Counts Per Inch
FPS	Frames Per Second
IPS	Inches Per Second
EEPROM	Electrically Erasable Programmable Read-Only Memory
SRAM	Static Random Access Memory

1 Introduction and Motivation

It is estimated that lower limb amputations account for up to 86% of all amputations [1,2], with 113 000 new lower limb amputations annually in the United States and 40 000 – 60 000 in Germany [1,3]. In the United States, the majority of cases are due to vascular disease (54-86%) and trauma (16.45%), with the remainder due to cancer or congenital problems [2–4]. The majority of amputations are performed on males or those over 50 years of age [3]. While amputation can cause various psychological issues, the most direct impact is upon mobility.

Lower limb amputees are typically fitted with an appropriate prosthetic in the weeks or months following their surgery [5–7]. Comfort, function, durability and cosmetics are all taken into consideration. The importance of a well-fitted prosthetic cannot be understated as it directly impacts the amputee's mobility and quality of life. A poorly-fitted prosthetic will not only be uncomfortable and tiring to use [6,8,9], but the resulting restriction upon physical activity may also reduce the amputee's ability to return to work, maintain social relationships, and participate in leisure activities [10]. The satisfaction with and quality of the prosthesis depends greatly upon the prosthetist's experience and expertise [11–18]. Various methods of evaluating fit are available, but they are either subjective, depending upon patient feedback and technician experience, or not suitable for clinical use due to size and cost of equipment, additional required expertise, cumbersome use or exposure to radiation. Faulty evaluations can lead to inappropriate and expensive adjustments or replacements of the prosthesis [19].

There exists a need for more objective, quantitative methods of assessing socket fit [18]. Pistoning – the vertical motion of the residual limb inside the prosthetic socket – has long been considered characteristic of poor fit [16,20–25]. However, no threshold of acceptable motion has been agreed upon [20] nor has the correlation between amount of pistoning and patient experience received much attention.

A project at the Institute of Mechatronic Systems at the Technische Universität Darmstadt has been underway for several years with the goal of creating a simple, cost effective system to evaluate pistoning in a clinical setting and allow such studies to be more easily conducted. The developed system utilizes optical sensors usually used in computer mice to measure the relative motion between liner and socket wall through small hole. The system was first developed by an Advanced Design Project (ADP, July 2015) [26] with an experimental investigation of favorable configurations as the topic of a bachelor's thesis (September 2016) [27]. The description of the system and its potential was published in October 2016 [18]. Assessment of the system's performance in gait situations was performed by an Advanced Research Project (ARP, November 2016) [28], with further experimental validation of the system being the topic of a second bachelor's thesis (April 2018) [29].

These works identified various shortcomings in the system's hardware, software, and communications structure. The purpose of this thesis is threefold: firstly, optimize the system's hardware, software, and communications; secondly, expand its capacity to at least four sensors as was originally

proposed; and thirdly, quantify the final system's accuracy and precision. Upon completion of the work, the system will be ready for use in a pilot study.

Chapter 2 will cover background information relating to prosthetics and relative motion between residual limb and socket, the operating principle of the optical sensor used, and serial communication. The proposed measurement system and experimental results to date will be presented in Chapter 3.1. As most previous work is published only in German, this chapter will provide more detail than would ordinarily be appropriate. The selection and evaluation of the new microcontroller will be described in Chapter 4. Chapter 5 will detail the software optimization process and its results, and Chapter 6 will then examine the improved system's performance on a test rig. A description of the finalized measurement system with sensor array will take place in Chapter 7, with concluding remarks to follow in Chapter 8.

2 Background Information

This chapter will first discuss anatomical terminology and the human gait cycle in Section 2.1 before providing an introduction to lower limb amputations, prostheses and the importance of pistoning in Section 2.2. The operating principle of the sensor chosen for the measurement system will be presented in Section 2.3 and an overview of the transmission methods used in the system given in Section 2.4.

2.1 Terminology and Gait

Although this thesis will not be focusing on the human body, a familiarity with basic terminology and the human gait cycle will be useful for understanding the discussion of amputations and prostheses in the subsequent section.

2.1.1 Anatomical Terminology

Three reference planes are used to divide the body (Figure 2.1) into front and back (coronal or frontal plane), left and right (sagittal plane), and upper and lower (transverse plane). A location within a body part may be described as anterior or posterior (towards the front or rear), proximal or distal (towards or away from the rest of the body), and medial or lateral (towards or away from the body's midline) [30]. Directions may correspondingly be given as anteroposterior (ap), proximal/distal (pd), and mediolateral (ml). These designations indicate the direction of travel, e.g. from anterior to posterior, and may therefore be reversed (compare with Figure 2.2).

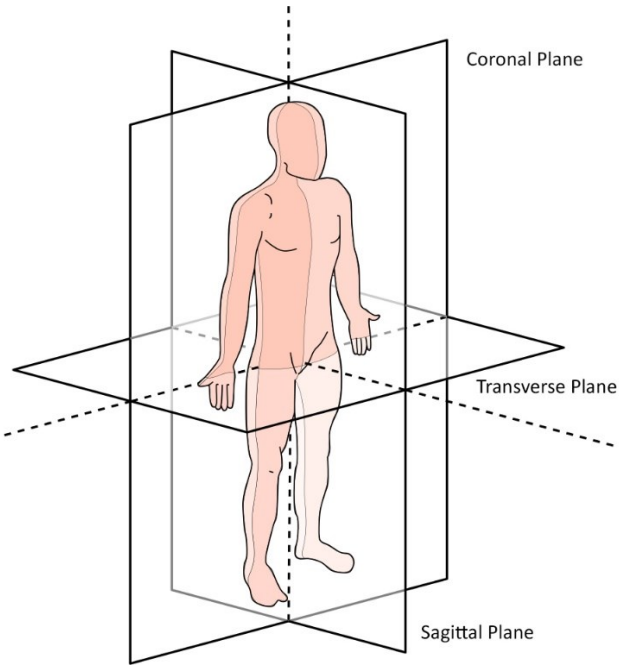

Figure 2.1: Anatomical planes, adapted from [31].

The human leg, exclusive of the foot, has of four bones: the femur (thigh bone), the patella (knee cap), the tibia (shin bone), and the fibula (calf bone). As their colloquial names suggest, the tibia and fibula are located to the anterior and posterior of the leg, respectively.

The leg may perform three distinct types of motion (Figure 2.2). Flexion and extension are the opening and closing of a joint in the sagittal plane. Bending the knee is a flexion, straightening it an extension. Adduction and abduction take place in the frontal plane and describe the motion of the distal end of an element towards or away from the body's midline relative to its proximal end. Lifting the leg out to the side is abduction, bringing it towards the other leg adduction. Finally, internal and external rotation, alternatively known as medial and lateral rotation, is movement in the transverse plane. An internal or medial rotation describes rotating the body part inwards (e.g. turning the foot in) while an external or lateral rotation describes rotating the body part outwards (e.g. turning the foot out) [30].

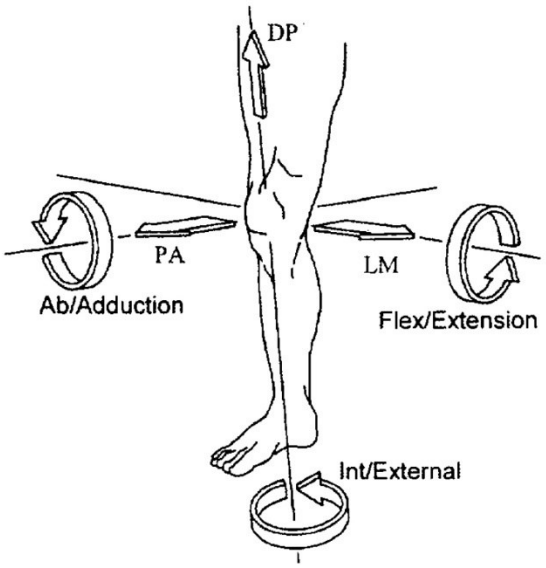

Figure 2.2: Motions of the lower leg and anatomical directions [32].

2.1.2 The Human Gait Cycle

During ambulation, each leg performs a repeating sequence of motions known as the gait cycle. Colloquially, one cycle is the equivalent of a stride, with each stride consisting of two steps. The gait cycle consists of a stance phase, when the foot is on the ground, and a swing phase, when the foot is in the air. They are generically considered to take up 60% and 40% of the cycle, respectively, although the exact proportions depend upon walking velocity [8,30]. Seven major events occur every cycle:

1. Initial contact
2. Opposite toe off
3. Heel rise

4. Opposite initial contact
5. Toe off
6. Feet adjacent
7. Tibia vertical

These events mark the transitions between the different periods of the two phases:

1. Stance Phase:
   i. Loading response
   ii. Mid-stance
   iii. Terminal stance
   iv. Pre-swing
2. Swing Phase:
   i. Initial swing
   ii. Mid-swing
   iii. Terminal swing

The gait cycle, along with the major events and periods, is illustrated in Figure 2.3.

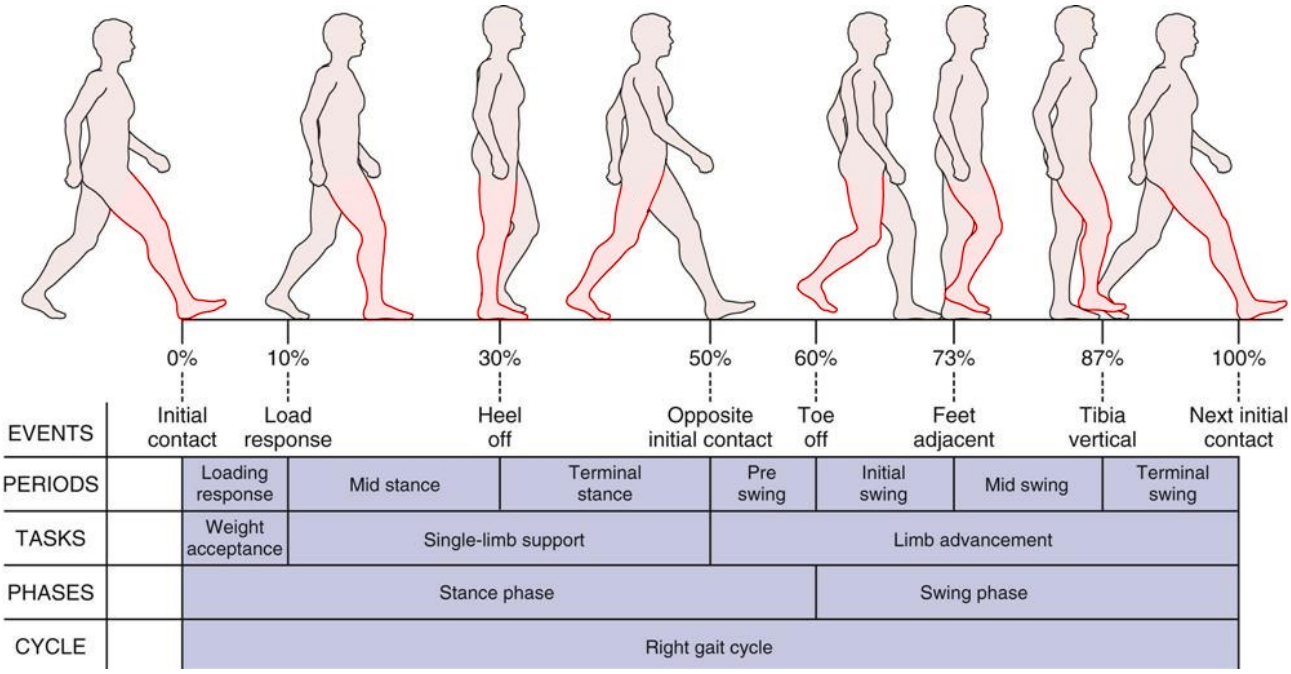

Figure 2.3: The human gait cycle, shown by example of the right leg. Adapted from [33].

The hip flexes and extends once per cycle, with the maximum degree of flexion occurring in the mid-swing and the maximum degree of extension in the terminal stance or pre-swing phases. The knee experiences two flexions and extensions each cycle. The first extension occurs just before initial contact with the corresponding flexion taking place during the loading response and mid-stance phases. The second extension then occurs during the late mid-stance, the accompanying flexion during the initial swing.

The leg undergoes both a pendular and a rotational movement. The pendular movement is easy to see, while the rotation is more difficult. Rotation occurs primarily in the hip and knee joints in the

coronal plane (as opposed to the ankle). Peak external rotation occurs at the beginning of the initial swing and peak external rotation at the end of the loading response [8]. Put more simply, the limb rotates externally during stance and internally during swing.

The ground reaction forces that occur during stance phase are shown in Figure 2.4 for normal and transtibial amputee gait. The two peaks visible during normal gait during loading response and terminal stance, i.e. when weight is first being accepted onto the limb and directly before the limb pushes off for the swing phase. These two peaks are not present for the transtibial amputee gait. Instead, the reaction force is greatest after loading response and slowly declines until the terminal stance. Of particular interest for the question of relative motion within the prosthetic is that the magnitude of the motion corresponds well with the magnitude of the reaction force [34], as seen in the right of the figure. Pistoning will be discussed in greater detail in Section 2.2.2.

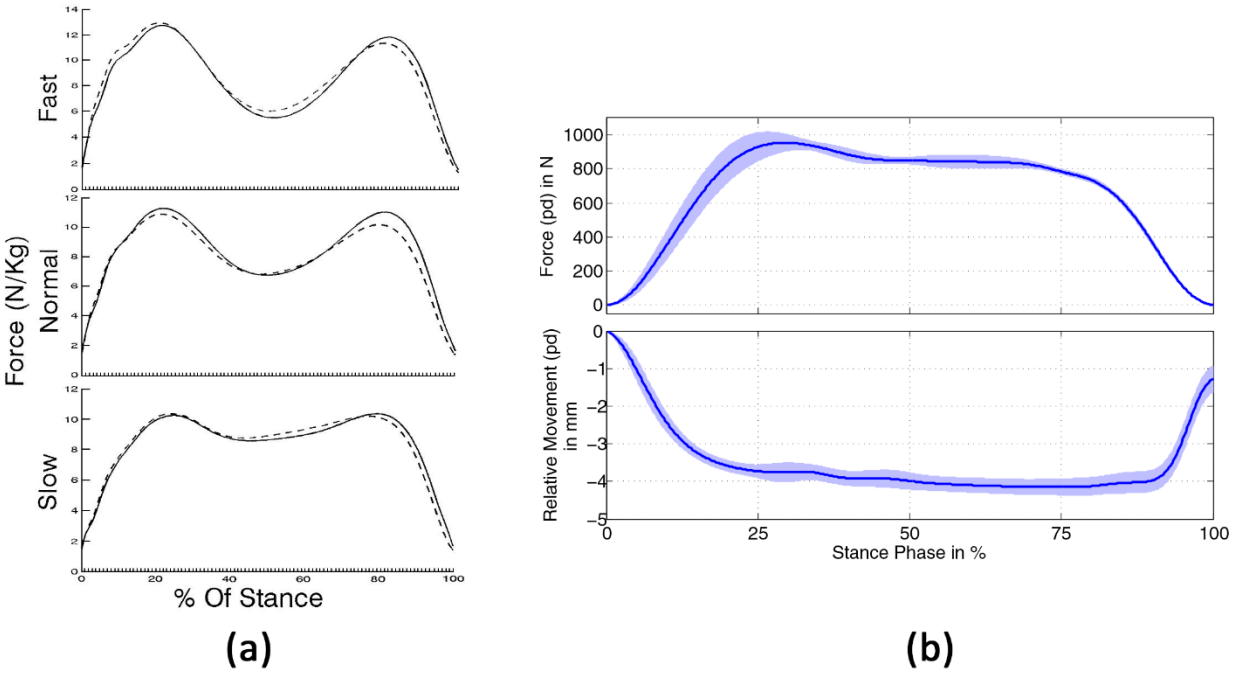

Figure 2.4: Ground reaction forces for normal gait in N/kg (left) [35] and for transtibial amputee gait in N (right) with corresponding relative movement in mm [34]. Plots are shown over stance phase only.

2.2 Lower Limb Amputations and Prosthetics

This section will discuss the different levels of lower limb amputation, the types and structures of prostheses, as well as their fitting and fabrication. Following this, the relative motion between socket and residual limb will be discussed. The phenomenon and its consequences will be explained in more detail and a summary of the existing measurement systems and displacement magnitudes given.

2.2.1 Overview

Lower limb amputations are distinguished based upon their level: foot or ankle, transtibial (TT), through the knee, transfemoral (TF), and hip (Figure 2.5) [5,6,36]. Foot, ankle, knee and hip amputations can be classified either by location (e.g. transphalangeal or transmetatarsal for the foot) or type of surgery (e.g. hip disarticulation or hemipelvectomy). Transtibial and transfemoral amputations can be described by the remaining length of tibia or femur. The length and surface geometry of the residual limb influence both the health of the limb and the potential for a successful prosthetic fit. For transtibial amputations, lengths of less than 20% or greater than 50% are not desirable due to a small moment arm for knee extensions and poor blood supply at the distal end, respectively. Within these restrictions, at least 8 cm of bone is considered ideal [6]. For transfemoral amputations, lengths shorter than 35% are uncommon. For good prosthetic fit, the limb should measure at least 10-15 cm from the groin and end at least 10 cm above the distal end of the femur to allow room for the prosthetic knee [6].

Figure 2.5: Amputation levels; left to right: foot, transtibial, knee, transfemoral, hip [36].

The sophistication and appearance of a particular prosthesis depends upon the amputee's individual needs and tastes, but the basic structure is the same: a foot and ankle, a pilon, a knee (in the transfemoral case), and a socket [30]. The socket is made of rigid plastic [5,6,13,15] and several designs differing in load distribution and suspension options are available to accommodate different lifestyles. The two most common transtibial options are the Patella Tendon Bearing (PTC) and the Total Surface Bearing (TSB) types. PTB sockets are the most common type [5,37] and shift pressure away from sensitive areas (e.g. bony areas) onto muscles and tendons, specifically the patella tendon [11]. In contrast, TSB sockets distribute pressure evenly across the entire stump [5,11,38].

Transfemoral sockets are of either the Quadrilateral or Ischial-Containment types [6], the difference being the degree of pelvis inclusion and additional support that configuration brings. Examples of prosthetics for different levels of amputation are shown in Figure 2.6.

Figure 2.6: Prosthetics for transtibial amputation (left), transfemoral amputation (center), and hip amputation (right) [39].

The socket is held onto the residual limb with a suspension system, of which there are three main types: shuttle lock, suction, and vacuum. In a shuttle lock system, a pin at the distal end of a padded liner (Figure 2.7) is inserted into the shuttle lock at the bottom of the socket [5,40]. This system is recommended for older amputees or those with reduced mobility, as it is the easiest to don and doff (put on and take off), but also offers the least amount of comfort and proprioception [17,40]. Suction and vacuum systems both operate by creating adhesion between limb and socket with lower pressure. In suction systems, this effect is achieved passively when air is expelled through a one-way valve upon insertion of the limb; in vacuum systems, a pump is used to remove the air. Both require a TSB socket and take longer to don and doff but result in better performance than shuttle lock systems [40]. A sleeve is used with these suspensions to ensure a good seal.

Figure 2.7: Liner with pin for shuttle lock suspension [40].

A protective liner is worn over the limb. Its purpose is to maximize comfort and stability by ensuring good adhesion between limb and socket, which reduces pistoning [6,17,41]. Different materials offer properties to suit each individual's suspension type, activity level, and personal preferences. Liners typically have an inner layer of silicone, polyurethane or copolymer and a textile outer layer.

In fabrication, a high degree of customization is needed to meet both functional and comfort requirements. An accurate evaluation of the limb is essential for a well-fitting prosthetic [13], good performance of the prosthesis is highly dependent upon the knowledge and skill of the technician [11–15,17]. Necessary skills include the ability to accurately evaluate the mechanical behavior of the limb, understand correct limb alignment, and the design, modification, and fabrication of sockets. The technician's influence is so great that studies use a single individual for all their custom prostheses to keep inter-technician variability out of play.

Traditional socket fabrication involves manually casting the limb with chalk bandages or plaster. While doing so, the technician applies pressure in loading zones (e.g. patellar tendon area for a PTB socket) and accommodates sensitive areas (e.g. the distal end) [11,13,14]. A plaster model of the limb is then made and compared to the patient's measurements. Manual adjustments are performed as needed. Once the model is satisfactory, a check socket, often of clear plastic, is fabricated [13,14]. The result is evaluated and modifications made; this process is repeated until the final form is reached [11].

A "new paradigm" [13] proposes to shift most of this iterative process to the PC. Exterior and interior limb geometry is acquired in 3D with laser scans and a CT or MRI scan. The unloaded, unstressed limb is reconstructed in CAD using reverse engineering principles. The designer then models the socket around the limb, paying attention to surface details to avoid potentially sensitive areas such as lumps or scars. The design is validated first with a parametric 2D model and later with more extensive 3d physics-based simulations. Physical prototypes to test and validate the design in practice are made with rapid prototyping technology. The goal of this process is to reduce the number of positive models, require less amputee involvement, and produce more accurate sockets. It is also an attempt to reduce the dependence on the expertise of individual technicians. It does not, however, address the lack of quantitative means to evaluate the quality of fit of the physical prototypes.

2.2.2 Relative Motion in a Prosthesis

Pistoning is the vertical displacement of the residual limb within the prosthetic socket. It can occur in any of three layers: socket-liner, liner-skin, and skin-bone (i.e. within the soft tissue) [42]. This motion is due to the natural forces at play during ambulation, but can be exacerbated several factors, including volume loss in non-suction or vacuum sockets (4-10% over the course of a day [25]) and by a poorly-fitting prosthesis [20,21,24,25]. Even in a well-fitted prosthesis with good adhesion (slippage between skin, liner and socket is minimal), displacement can still take place in the soft tissue [15,43]. In such cases, the amount of subcutaneous fat and muscular coverage of the bone, as

well as limb length, are important for stability [42,44]. Although a tight fit decreases pistoning and increases stability, too tight a fit increases interface pressures and shear stresses, which can also be problematic [16].

Depending on its severity, pistoning can impact the patient in various ways. A loose fit can make the connection between limb and prosthesis feel unstable, which leads to a loss of proprioception and kinesthesis, increased energy costs for ambulation, increased risk of falls, and, in extreme cases, the accidental exiting of the limb from the socket [16,19,25,43]. Relative motion between skin and socket can damage the soft tissue: discomfort and pain, skin damage, blisters, edema, osteomyelitis, dermatitis, lacerations and necrosis have all been documented [16,19,21,43,45]. Minor complaints discourage prolonged use of the prosthesis due to the associated discomfort [24], while more serious ones mandate immediate discontinuation of use [6,45]. Predisposition to osteoarthtritis is an additional risk [19]. Despite general agreement that the degree of pistoning is an indication of fit quality [13,20,21,23,46], there is no agreed upon standard for what amount of motion is acceptable for a „good fit“ [20] nor is it clear if or how much this threshold may vary between patients.

Numerous studies have been conducted to measure the relative motion between the residual limb and the socket. These studies can be broadly categorized by method: radiographic studies employing videofluoroscopy [19], radiographs [21,24], DRSA (biplane dynamic roentgen streoegrammetric analysis) [12], Roentgen stereophotogrammetry [42], and SXCT (Spiral X-Ray Computed Tomography) [17,21]; and non-radiographic studies employing ultrasound [24,47], photography/videography [17,21], and motion capture [34]. Measurement results are not always directly comparable due to different experimental methodologies and socket-suspension-liner configurations but do establish an expected range of motion. For transtibial amputees, values between socket and liner or socket and skin typically range between 2-9 mm, although values as high as 41.7 mm have been recorded [12,17,24,25,34,37,44–47]. Motion between socket and tibia is higher, with typical report values in the 20-40 mm range [15,25,42–44,46,48].

Rotation between limb and socket is a further source of relative motion. It is caused by the natural rotation of the limb during the stance phase [40,42]. Radiographic methods have been used to measure the rotation between socket and tibia. Results ranged from 1-14° depending upon socket and suspension type, situation (e.g. walking on flat ground or ascending stairs), and measurement method [15,22,23,42,48].

Although each of the measurement systems mentioned has been proven capable of producing the desired measurements, most are not appropriate for routine clinical use. Radiographic methods are mostly suited only for static, simulated load conditions and should be used sparingly because of radiation exposure concerns. Many of the approaches require expensive machines, special laboratory conditions and setups, tedious – sometimes manual – data evaluation, and experienced operators.

2.3 Optical Sensors

Optical sensors of the type used in the functional model consist of a light source (an LED or laser), a lens, a CMOS (complementary metal-oxide-semiconductor) detector, and a signal processing unit. The light source illuminates the surface at an angle, casting the texture of the micro-surface into sharp relief [49]. The lens directs the emitted light towards the surface and focuses the reflected light on the detector. Motion is determined by comparing two subsequent images, or frames, obtained from the detector. Since comparing the full image would be too computationally taxing – sensor resolutions can range between 15×15 px to 30×30 px [49–52] – a small e.g. 5×5 px window from the center of the second image is overlapped and matched to the first. The chip evaluates how well the window matches each of the full-image pixels and, once a best overlap is found, compares the surrounding layer of pixels to confirm [49]. The relative X and Y displacements are then determined in counts, which are read by an external microcontroller. A schematic of the system is shown in Figure 2.8.

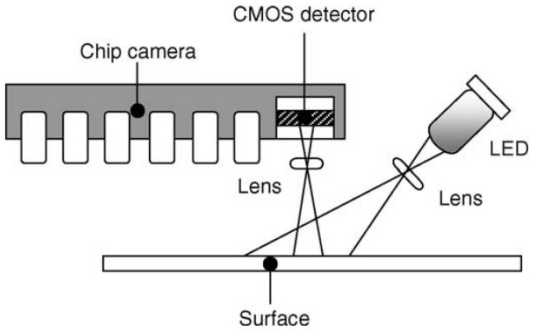

Figure 2.8: Optical sensor diagram [49].

Experimental evaluations of this type of sensor have revealed some valuable insights. They are not suited for transparent or reflective surfaces [49]. Relative errors have been measured to less than 0.8 mm over a 50 mm range [53] and 0.1% over a 160 mm range [50]. Accuracy depends upon the path and orientation of the sensor relative to the surface. Diagonal motion showed poorer performance than single-direction linear motion; a proposed explanation is that the shape of features seen during diagonal displacement is not a necessarily a linear combination of the X and Y displacements, making it more difficult for the image processing algorithm to correctly identify the distance travelled. This effect is even more pronounced for circular paths [53]. Sensitivity has been shown to be greater in X direction than Y [49,53]. In both directions, it has been shown to depend upon the sensor's orientation relative to the surface, which indicates that the surface texture interferes with consistent detection [50]. Sensitivity also decreases when either the offset between sensor and surface or the velocity exceed the recommended range; in both cases, sensitivity slowly drops off until the sensor ceases performing reliably [50].

A discussion of the specifications and previously evaluated performance of the functional model's sensor will be presented in Chapter 3.

2.4 Synchronous and Asynchronous Serial Communication

The optical sensor and microcontroller communicate via Serial Peripheral Interface (SPI). SPI is synchronous, meaning that transmissions are synchronized to a clock signal and multiple data transmission lines are used. A master device (the microcontroller) controls one or more slave devices (the sensors). The master determines the frequency of the clock and transmits an oscillating signal to each slave over the Serial Clock (SCLK) line (slaves must be compatible with the chosen frequency). The master sends data over the Master Out/Slave In (MOSI) line and receives it via the Master In/Slave Out (MISO) line. Each slave has a Slave Select (SS) line used by the master to "wake up" the slave and permit it to receive and transmit data. This means that only one slave is active at any given time and that all devices can share common SLCK, MOSI, and MISO lines. This is the normal SPI structure. The alternative is to daisy-chain the slave devices together. Here, the MOSI line from the master goes to the first slave. This slave's MISO line is connected to the next slave's MOSI, and so on down the line. The final slave device's MISO line feeds back to the master. All slaves share a single SS line. Instead of travelling directly between master and slaves, bits are shifted from one slave to the next until they finally reach the master. It is better suited to output-only situations and requires the master to send enough bits to shift through all the slaves [54]. Both the normal and daisy-chain structures are shown in Figure 2.9.

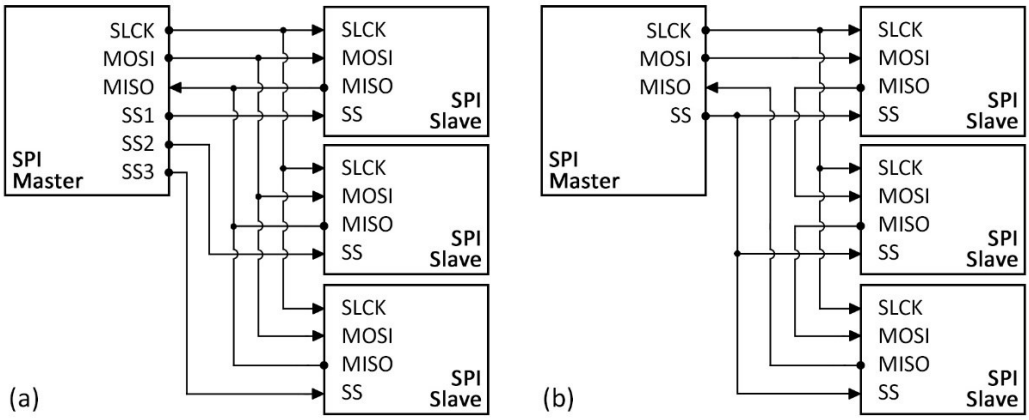

Figure 2.9: SPI structures: (a) normal, (b) daisy-chained.

The system may be configured to send data with the most significant bit first (MSB) or least significant bit first (LSB), and to sample bits on either the rising or the falling edge of the clock signal. The configuration is often determined by the devices involved. SPI can reach faster speeds than asynchronous serial, in part because there is no need for start or stop bits thanks to the predictability of the clock; however, SPI requires a fixed command structure. SPI can support many slave devices, but at the cost of additional signal lines.

The microcontroller and PC communicate over asynchronous serial. Both devices must be configured to use the same baud rate (the transfer speed, expressed in bits per second or bps). Although higher baud rates increase transmission speed, many devices cannot exceed 115200 bps. Since

transmissions are not tied to a clock signal, additional synchronization bits are needed: one start and one to two stop bits. There is also the option of adding parity bits as a form of simple error checking. In contrast to SPI, asynchronous serial requires only two wires – one for transmitting, one for receiving. Different interfaces are possible: full-duplex (both devices are able to communicate simultaneously), half-duplex (devices must take turns communicating), and simplex (one-way communication with only one wire required).

3 The Proposed Measurement System

The proposed system has been in development for a number of years. This chapter will first present the system in its current state, then summarize the experimental results to date in order to establish a foundation for the work done in this thesis. Since one of the assigned tasks is to optimize communication and software performance, special attention will be given to difficulties encountered in the past.

3.1 System Requirements and Overview

System requirements were presented in great detail by the ADP group [26] and the most important of these will be summarized here. The purpose of the proposed measurement system is to measure the relative motion between skin or liner and prosthetic socket for both transtibial and transfemoral amputees. The system must be compatible with the most common suspension systems and be independent of residual limb geometry. It must be able to capture all six degrees of freedom – translation in the ap-, pd-, and ml-directions and rotation around the ap-, pd-, and ml-axes – and be used in conjunction with existing motion-capturing and gait analysis systems. It must also be capable of measuring motion in a variety of gait situations, including on the treadmill, on staircases, and in every-day situations. Although intended for laboratory use, the possibility of employing the system outside the laboratory is an important consideration. For measurements to accurately reflect the relative motion, the system must either impact the patient's gait in a calculable, predictable manner or, ideally, not at all. The maximum weight to be carried by the subject was set at 10% of their body weight with the weight of components affixed to the leg not to exceed 100 g.

Given the wide variety of sockets, suspensions, liners, and residual limb geometries, calibrating the system is essential for achieving accurate results. A maximum initial calibration duration was set at 60 minutes with adjustments during a measurement series to take less than 5 minutes. Duration of the measurements themselves was set at approximately 2 minutes. The system must have a resolution of at least 0.5 mm and be capable of measuring distances of at least 40 mm with a maximum absolute error of 1mm. The sampling frequency must be competitive with other available technologies, which ranges from 20-50 Hz for static measurements to 500 Hz for motion-capturing. Data collection should not require specialized knowledge and is ideally compatible with commonly used software solutions such as MATLAB. Data analysis should be automated as much as possible and require minimal specialized knowledge on the part of the evaluator.

After thorough research on possible measurement methods, the ADP group settled on optical sensors of the type outlined in Section 2.3. Details on the chosen sensor, the ADNS-9800, most commonly used in computer mice, will be given in Section 3.2.1. For greatest flexibility, the design was for a wireless system. A microcontroller is responsible for controlling the sensor unit, preprocessing data, and transmitting it to a PC; the NATIONAL INSTRUMENTS MYRIO [55] was recommended. To capture all degrees of freedom, two sensors each are affixed the anterior and lateral sides of the

socket. The subject carries a small box containing the microcontroller and a battery pack. PC-side collection and processing of the data is to be done in MATLAB.

A functional model was built to test the measurement system concept. Instead of the MYRIO, an ARDUINO UNO [56] was used. The Uno, however, has no wireless capability. In the functional model's first incarnation, no transmission to the PC took place. Instead, current, maximum, and minimum displacements were displayed on an LCD screen. The screen, paired with a pushbutton, also served to guide the user through the calibration process (this consisted of a choice of preset values or performing a calibration measurement). This model is shown in Figure 3.1. Later, the system was tethered to the PC with a USB cable to provide power and allow for data transmission.

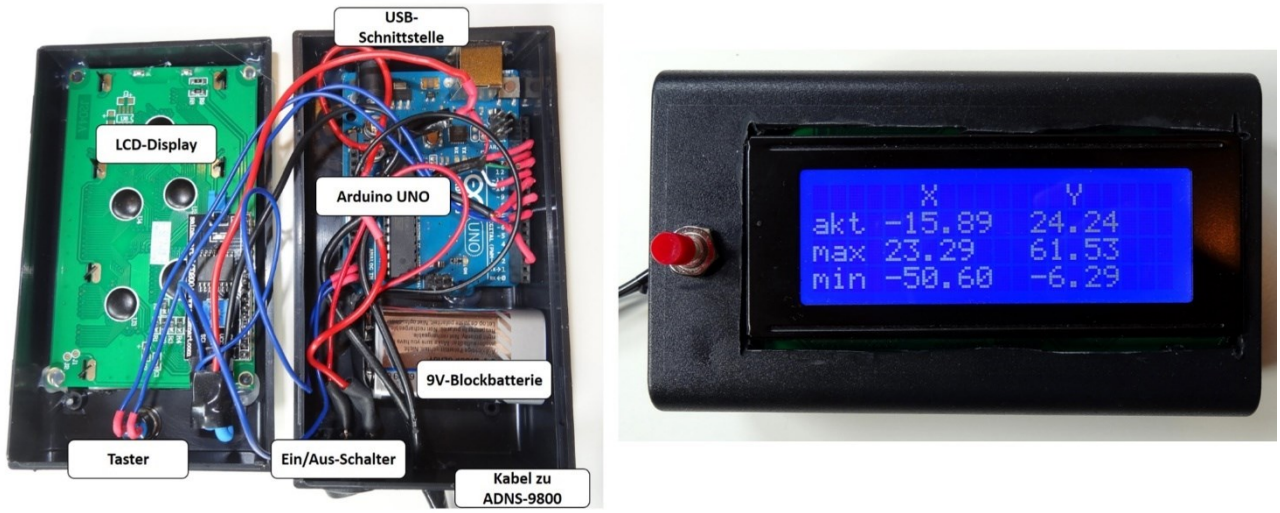

Figure 3.1: Functional model interior (left) and LCD display (right); adapted from [26].

3.2 Hardware

At present, the system's hardware consists of the sensor unit, the sensor unit's housing and mounting base, and the microcontroller. The most important aspects of all three will be touched upon here.

3.2.1 Sensor

Potential sensors were compared based on their resolution in counts per inch (cpi), maximum speed in mm/s, maximum acceleration in mm/s^2, framerate in Hz, required separation distance between sensor and surface, cost, supply voltage, form of data transmission, and availability. The ADNS-9800 by AVAGO TECHNOLOGIES [52] with its ADNS-6190 lens [57] was chosen due to its superior specifications, known price, and individual availability. A breakout board with both sensor and lens is available for purchase online from TINDIE [58]. With board, the unit's diameter is approximately 31.5 mm or 1.25 inches. A view from the bottom with lens attached is shown in Figure 3.2 (left).

The breakout board allows for easily changing the sensor's voltage supply between 3.3 V and 5 V. This is achieved by breaking the solder bridges for the old voltage and connecting the contacts for the new voltage (Figure 3.2 center). The voltage level of the microcontroller determines which voltage is appropriate.

The breakout board also provides convenient contacts for SPI. Headers were soldered to these contacts to eliminate the concern of loose or broken contacts and to allow for plug-and-play flexibility using DuPont connectors. Since SPI only requires one ground and the sensor has two (analog and digital), these two contacts (AG and DG) were soldered together. Pin designations are abbreviated due to space constraints (Figure 3.2 right): MI is MISO, VI is voltage in, SC is SCLK, and MO is MOSI. The MOT (motion) pin is not used in this application.

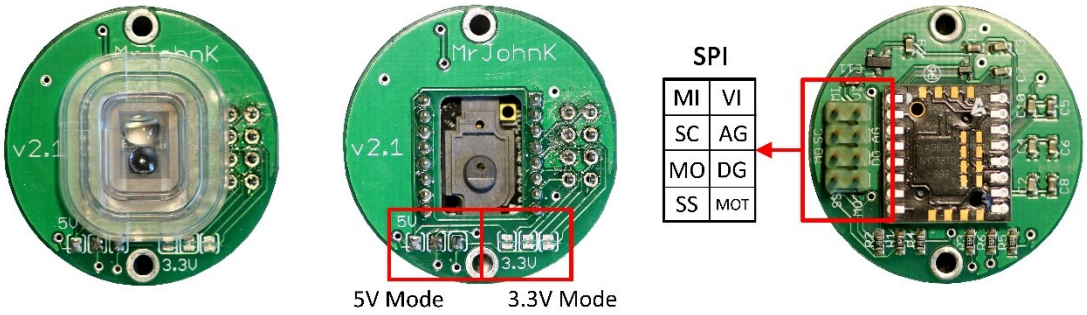

Figure 3.2: ADNS-9800 sensor, lens and breakout board. Bottom with lens (left), bottom without lens showing voltage selection contacts (center), and top with header pins soldered to SPI contacts (right).

The sensor has maximum speed and accelerations ratings of 150 ips (38100 mm/s) and 30 g (294.30 m/s^2) which far exceed the expected values of 71.4 mm/s (pd-direction) and 2572.7 mm/s^2 (ap-direction) for relative motion derived in [27] based upon data from [34]. The framerate can be chosen automatically by the sensor or set between 2000 and 12 000 frames per second (fps). The resolution can be set in between 200 and 8200 cpi in approximately 200 cpi increments. Both framerate and resolution are set programmatically by writing to the appropriate sensor registers. These specifications are summarized in Table 3.1.

Table 3.1: Key specifications of the ADNS-9800 optical sensor [52].

Characteristic	Value	Units
Supply Voltage	3.3 / 5	V
Max. Acceleration	30	g
Max. Speed	150	ips
Max. Resolution	50 – 8200	cpi
Communication	SPI	
Unit Price	12 – 25	€

The X and Y displacements are given in counts, which can be transformed into a distance using the set resolution and the appropriate conversion factor. In practice, a surface-dependent calibration factor is also necessary to correctly scale the result.

The sensor also registers a SQUAL (Surface quality) value that indicates the number of valid features seen by the sensor in the current image; the number of valid features is equal to four times the SQUAL-value. The SQUAL-value ranges between 0-169. Fluctuations between frames are to be expected, particularly when the sensor is in motion. A series of sample values acquired while moving over white paper is provided by the sensor manufacturer and is reproduced in Figure 3.3

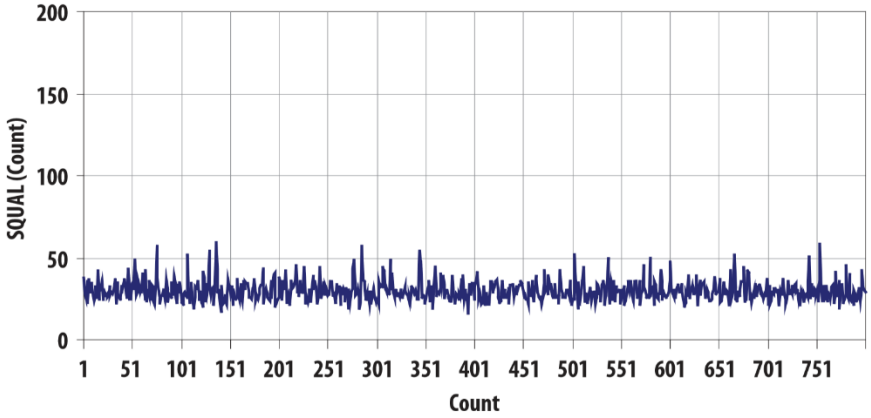

Figure 3.3: Sample SQUAL-values obtained from a measurement over white paper at 6 inches per second with 2.4 mm separation between surface and sensor [52].

3.2.2 Sensor Housing and Attachment Method

A housing was designed to protect the sensor unit and ease its attachment to the prosthetic. It consists of two parts: the PCB board nestles into the bottom half, which has a cutout for the lens; the top half has a hole for the cable and protects the entire unit. Mounting points on the socket are levelled to ensure a good contact area between housing and socket and to ensure that the sensor sits tangential to the underlying liner's surface. The socket wall at these sites may need to be thinned to ensure the optimal separation distance between sensor and surface, given to be 2.4 mm [52]. A small hole is drilled in the center to allow the sensor access to the liner. Since modifications to the socket are necessary to prepare these locations, a test socket will need to be manufactured. Once the mounting site is prepared, a placing guide is inserted into the hole and a mounting base glued onto the socket (Figure 3.4 left). To maintain pressure conditions on suction and vacuum sockets, two seals are placed between the lens and the socket wall. Three screws run through the housing into the base and hold everything together. An exploded view of the housing is shown in Figure 3.4 (right). This system was proposed by the ADP group and has remained unchanged since, although it

has not yet been fully put into practice. It achieves a firm attachment and thus minimizes play, which would contribute to measurement errors.

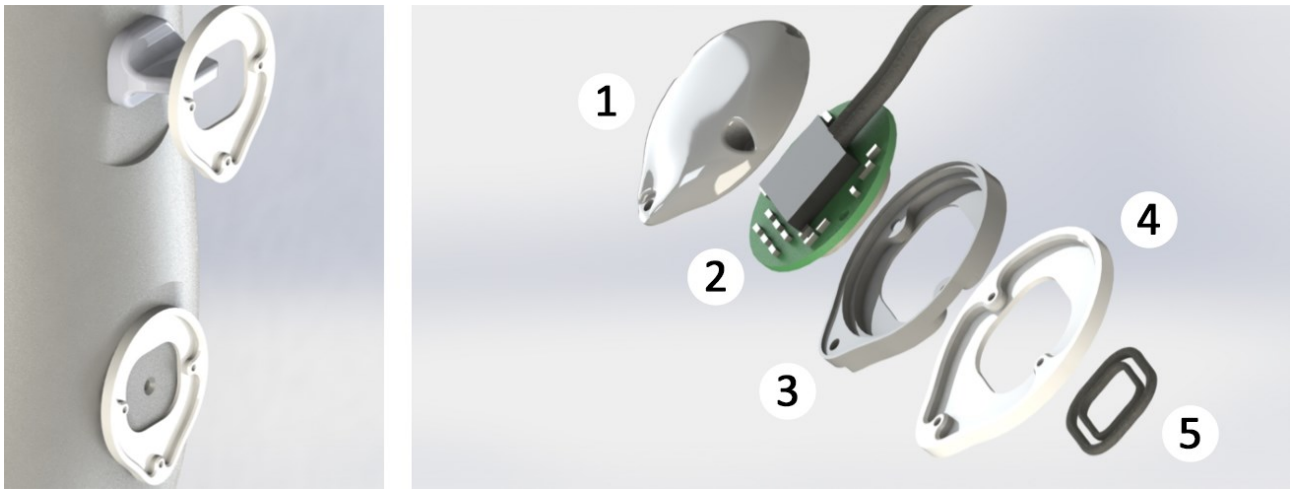

Figure 3.4: Placement guide and mounting base during and after attachment (left), exploded view of the unit (right): housing top (1), sensor (2), housing bottom (3), mounting base (4), and seals (5). Adapted from [26].

3.2.3 Microcontroller

When designing the measurement system, the ADP group considered two microcontrollers: the MYRIO by NATIONAL INSTRUMENTS and the ARDUINO YUN. They recommended the MYRIO due to its excellent specifications and availability at the institute. However, the functional model used an ARDUINO UNO, a widely available entry-level microcontroller. A further benefit of the UNO was that a sample ARDUINO code provided by the manufacturer was made publicly available on GITHUB [59], which allowed for the system's potential to be easily and quickly evaluated.

Since one of the tasks of this thesis is to find a more suitable microcontroller, an in-depth discussion of the requirements and specifications of various models, including the UNO and MYRIO, will be undertaken in Chapter 4.

3.3 Software

A structural overview of the Arduino code and MATLAB Graphical User Interface (GUI) provided as a starting point for this thesis will be presented here with a more detailed treatment to follow in Chapter 5. These codes will be referred to as the original versions in later chapters and are provided in the Digital Appendix.

3.3.1 Arduino Sketch

Arduino programs are referred to as "sketches." Sketches are typically developed in the ARDUINO IDE (Integrated Development Environment) and are written in C/C++. They must contain the functions

`setup()`, run once upon startup, and `loop()`, run continuously thereafter. A sketch may be split into different files within the parent folder for better oversight; all files in the folder are uploaded to the device. This is useful when the code is long or if e.g. firmware for an external component is to be included.

The student work adapted and expanded one of the two manufacturer provided sketches ("ADNS9800test-serial.ino" and "ADNS9800testPolling.ino" [59]). Although there have doubtless been numerous intermediate versions, the discussion in this thesis will restrict itself to one the two versions provided at the start of work[1], "Laser_Mouse_Matlab_v03b." Although these are the most recent versions, their exact context is unclear. Statements made about them can therefore not be generalized to all versions of the sketch used during previous work.

The sketch begins by defining the sensor registers, the necessary libraries to support SPI, AVR/PGMSPACE (for uploading the sensor's firmware), and the LCD screen, and variables. It then initializes serial communication to the PC and configures the SPI communication to the sensor. The sensor is initialized as outlined in the datasheet, including firmware upload. Six registers (Product_ID, Inverse_Product_ID, SROM_ID, Motion, Configuration_I and Configuration_II)[2] are read and transmitted to the PC in binary and hexadecimal formats. Once the sensor is initialized, the program enters calibration mode. The user may choose to use the last used calibration factors stored in EEPROM (Electrically Erasable Programmable Read-Only Memory), from a list of hard-coded presets, or to perform a calibration measurement. If measurement is chosen, the LCD screen will direct user to move the sensor 50 mm in the X and Y directions individually and then use the obtained values. Once a calibration option has been selected, the measurement process begins. The sensor registers containing the X and Y displacement counts and the SQUAL-value are read. The microcontroller converts the displacement from counts to mm, scales these values with the chosen calibration factor, and adds them to the previous value to arrive at the absolute displacement. It also checks if the new value represents a maximum or minimum, in which case the appropriate variable is updated. Current, maximum and minimum X and Y displacements, the SQUAL-value, frame number and timestamp are sent as ASCII characters over the serial port to the PC at a predetermined frequency. The frame number is generated by the microcontroller and is not a true reflection of the sensor's frame number.

3.3.2 MATLAB GUI

Following the ADP group's work in proposing the system and building the functional model, a MATLAB GUI was created to more easily collect the data sent by the microcontroller. As with the

[1] The other is "Laser_Mouse_Matlab_v04." There was no formal reason for working off one as opposed to the other aside from that the selected version worked immediately while the other did not.

[2] "Product_ID" is a unique identifier assigned to the sensor model; "Inverse_Product_ID" is its binary inverse. "SROM_ID" contains the revision of downloaded firmware. "Motion" indicates if motion has occurred, status of the laser, and sensor operating mode. "Configuration_I" defines the resolution, while "Configuration_II" defines frame rate mode (automatic or manual).

sketch, the GUI went through several iterations, its capabilities changing depending on what functionality was desired. The version titled “sensor.m” was provided at the start of this work.

The interface is shown in Figure 3.5. This version allows for easy data collection through a “Start/Stop” toggle button and the evaluation of percent errors over a given interval. This is achieved by entering the distance’s magnitude in the “delta” column and pressing the “Begin/End” button to record the initial and final positions registered by the sensor. These are displayed in the “begin” and “end” columns. The percent error is calculated automatically and displayed in the “error” column. All data associated with an instance of the GUI is saved to the location specified in “File name” when the measurement is stopped.

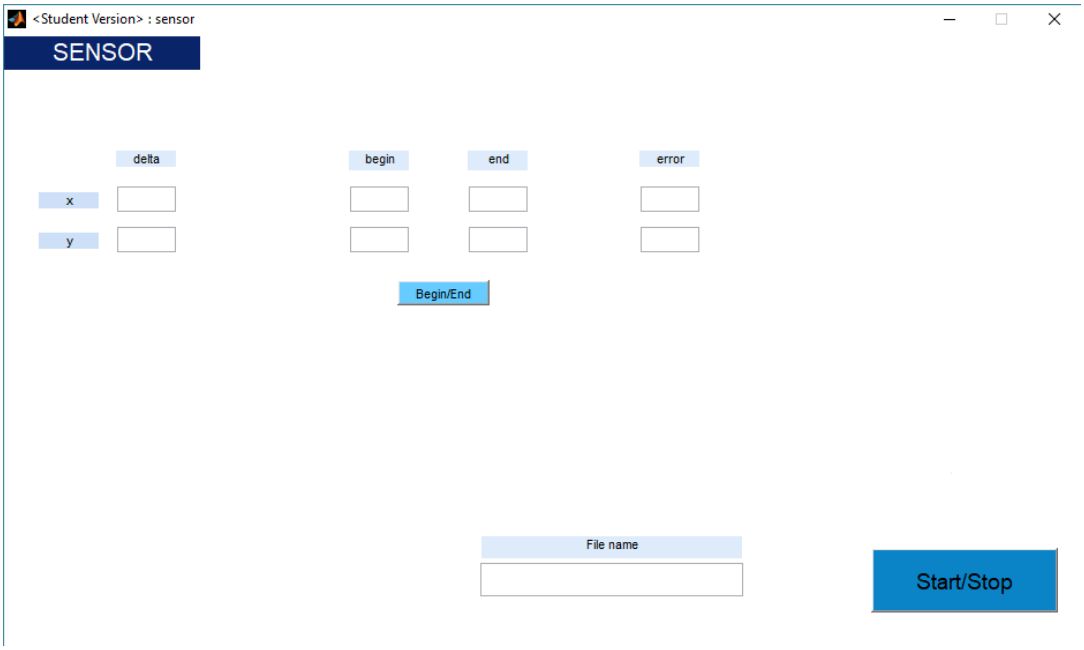

Figure 3.5: Appearance of the original GUI.

The code structure is also straightforward. Pressing the “Start/Stop” button initiates a while loop that runs until the button is pressed again. Each cycle receives one packet of data from the microcontroller. As the maximum and minimum displacements arrive, they are checked against the previously saved value and the largest (or smallest) of the two is saved. When the absolute X and Y displacements arrive, the program checks if an interval is currently being evaluated and either does nothing, records a starting point, or records an end point and updates a variable to indicate that a calculation is to be performed. After all data has been read and stored, this variable is checked and the percent error calculated if required. Finally, the visual elements of the GUI are updated.

3.4 Experimental Results and Improvement Potential

Initial, informal experiments performed by the ADP group using the first functional model indicated the system's potential. For a 100 mm distance covered at a speed of 300 mm/s, a maximum error of 0.75% was measured. The error increased for measurements on curved surfaces.

In his bachelor thesis, Somogyi investigated the influence of various factors on system performance in test rig experiments with the aim of identifying favorable settings [27]. Two test sessions using a two-stage fractional factorial screening design were performed. The first session evaluated the influence of distance covered, speed, calibration speed, resolution, and direction on five different liners (Figure 3.6). The two best-performing liners (materials 1 and 5) advanced to the second test session, which applied the settings identified as favorable in session one while testing the influence of distance from the surface and upper and lower hole diameters. The (-) and (+) values used for both test sessions are listed Table 3.2. A sampling frequency of 15.5 Hz was achieved with the baud rate set to 115 200 bps and the frame rate to 2 000 fps. For each combination of settings, 10 trials were performed in session one and 20 in session two.

Figure 3.6: Liner materials: Össur TF I-7032XX (1), Össur Comfort I-5406XX (2), Ottobock 6Y87-Skeo 3D (3), Ottobock 6Y85 - Skeo Skinguard (TD/AK) (4), medi Liner RELAX 3C/6C (5). Adapted from [27].

Since the analysis presented in the bachelor thesis was extensive – investigating the effects of factors and their interactions – only the major findings will be summarized here. Session one showed that the longer measurement distance, faster measurement speed, slower calibration speed, and higher resolution resulted in a smaller relative error. Only minimal differences were shown by direction. Session two showed the same performance for distance and speed, but a larger error in Y. A smaller distance between sensor and surface as well as larger hole diameters (both upper and lower) resulted in smaller relative errors. The conclusion was that only material 5 fulfilled the stated requirement of 1 mm absolute error over a 40 mm distance.

Table 3.2: Low and high values for the factors of the test sessions [27].

	Factor	Low (-)	High (+)	Units
Session One	Distance Covered	1	10	mm
	Measurement Speed	1	100	mm/s
	Calibration Speed	1	100	mm/s
	Resolution	200	8200	cpi
	Direction	X or Y		-
Session Two	Distance Covered	1	10	mm
	Measurement Speed	1	100	mm/s
	Direction	X or Y		-
	Distance from Surface	2.18	2.62	mm
	Upper Hole Diameter	4	21.5	mm
	Lower Hole Diameter	4	21.5	mm

The effect of zeroing frequency on accuracy and the meaning of the SQUAL-value were also examined. Frequent zeroing (e.g. every 1 mm) was associated with a greater relative error, and it was posited that the errors are more likely random than systematic and that they mostly cancel each other out over sufficiently large measuring distances.

A dependence between liner's texture and both the relative error and the SQUAL-value was noted in the first test session and was present again in the second (however, regularities for material 1 were no longer evident). The periodicity shown in both is attributed to the regular surface texture of the liner, and the differing degrees of expression (e.g. none for material one in session two) suggested that other factors are also at play. Plotting relative error against the mean SQUAL-value for all experiments revealed a broad scatter approximately following the shape of an exponential curve. It was concluded that the mean SQUAL-value could be used to estimate the relative error.

Experimental validation of the system's accuracy and precision on liner material 5 were done using a different test rig [18]. Three trajectories (linear X, linear Y, and diagonal) were tested at two velocities (100 mm/min and 2100 mm/min) over four distances (1 mm, 2 mm, 5 mm, 10 mm) with the number of repetitions varying by distance (45, 25, 10, and 5 respectively). The relative error for each trial was calculated and plotted with the standard deviation (Figure 3.7). The results showed negligible variation based on speed. Overall, accuracy and precision were slightly better in the X direction for both linear and diagonal paths. The anisotropic nature of the material may have contributed, although alignment errors between sensor and test bench cannot be discounted. For both directions, greater path lengths produced smaller errors. Errors ranged from 2 – 6% depending on the distance travelled.

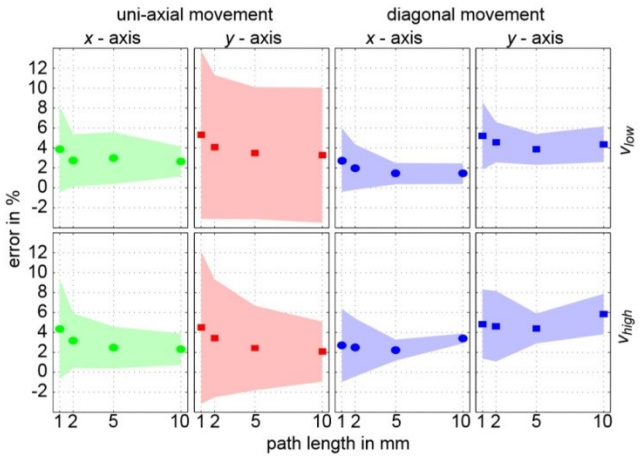

Figure 3.7: Mean (marker) and standard deviation (shaded area) of measurement error [18].

The ARP conducted measurements on unloaded knee bends and treadmill gait as well as preliminary experiments relating to calibration constants and the influence of a stretched liner [28]. The unloaded knee bends were performed at two frequencies (0.25 Hz and 0.38 Hz) by a seated subject moving their foot between two lines on the floor (the lines were two subject foot-lengths apart). A piece of liner 5 was attached to the shin and the sensor was elastically attached between foot and thigh. Pieces of scaled paper were affixed to both the shank and sensor. The motion was filmed so that sensor-derived data could be compared to that given by the relative motion of the papers. The sensor data was found to be within the precision of the camera and it was concluded that the sensor was suited for measuring motion over a curved surface.

The treadmill gait experiments utilized an orthosis. A piece of liner was strapped to the subject's calf while the sensor was attached to the lateral splint of the orthosis. A small hole drilled through the splint allowed the sensor to see the liner surface (Figure 3.8). Measurements were taken at three different speeds (0.6 m/s, 1.0 m/s, and 1.4 m/s) with the orthosis unconstrained or blocked at 30°.

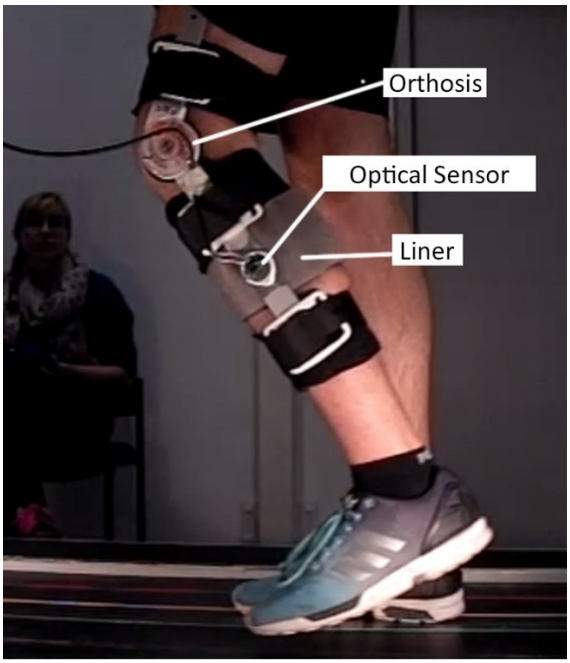

Figure 3.8: Profile view of the setup for the treadmill experiments. The orthosis is firmly strapped to the subject's leg, sensor integrated into the lateral splint, and liner wrapped around the calf. In translation from [28].

Twelve experiments were conducted with two subjects and qualitative differences between the two were discernable. Both ap- and pd-motion registered by the sensor displayed positive or negative trends with no obvious pattern being apparent. One such dataset is shown in Figure 3.9. Suggested possible explanations for the drift included the systematic accumulation of errors, differences between the calibrated and practically-required calibration factor, the orthotic slipping, or any combination thereof. Fourier and manual frequency analyses differed by 0.59- 2.96% with only one exception. The sensor curves qualitatively matched the expected knee angles, and both the times and magnitudes of the relative motion were within the expected range. Relative errors were estimated based upon the correlation of SQUAL-value and relative error found by Somogyi.

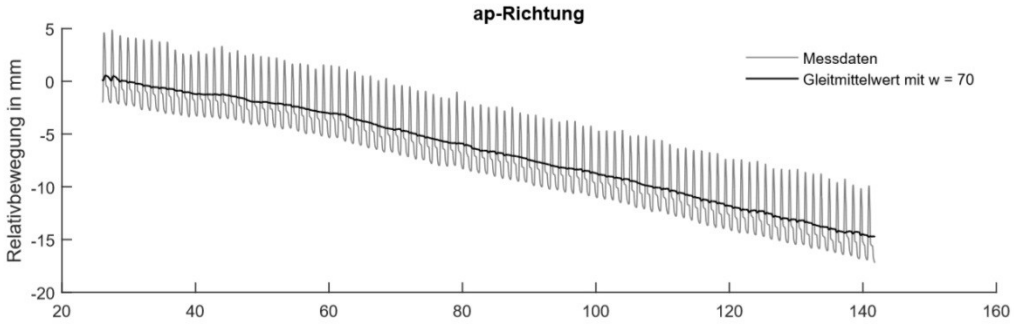

Figure 3.9: Motion in the ap-direction for blocked orthosis at 1.0 m/s with time in s on the x-axis and relative motion in mm on the y-axis; raw sensor data (light grey) and moving average with w=70 (dark grey) [28].

To estimate the dependence of calibration factors upon the stretching of the liner, three trials for varying degrees of stretch (0-20% in 5% increments) were performed over a 50 mm distance at a speed of 5 mm/min. The results show factors for the X direction to be greater than those of Y, with tendencies to increase and decrease with the stretch, respectively (Figure 3.10). To gauge the impact upon relative error, the SQUAL-values were again considered. It was found that, although the mean SQUAL-value for the different degrees of stretch differed, they were still within each other's standard deviations. Stretching was thus considered to have a negligible impact on accuracy.

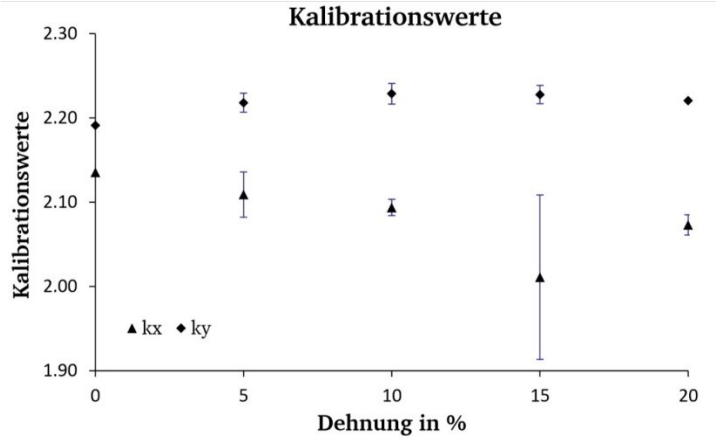

Figure 3.10: Calibration values measured over a 50 mm distance at 5 mm/min, three trials each marker; % stretch of the liner on the x-axis, calibration factor on the y-axis [28].

In his bachelor's thesis, Arnemann performed further experimental validation of the system in a representative study utilizing the same orthosis as the ARP [29]. Experiments were conducted in the Locomotion Laboratory of the Department of Human Sciences (Institute for Sport Science) at the Technische Universität Darmstadt using an instrumented treadmill and motion-capturing system. The existing functional model of the measurement system was used, running the same ARDUINO sketch and MATLAB GUI as were provided at the start of this thesis. As such, it suffered from the previously reported speed limitations and achieved transmission rates of only 30 Hz. No mention was made of the calibration process or factors used. The sketch was modified to accept a 5 V trigger signal to synchronize the various measurement systems.

The experimental setup of sensor, liner, and orthosis was the same as in the ARP, although here the liner was firmly attached to the calf with Velcro to ensure it did not slip. Ten motion-capture markers were placed upon the subject's leg (hip and ankle), the orthosis (center of upper splint, joint, directly above and below the sensor), on the sensor, and on the liner (two to the anterior of the orthosis, one to the posterior in a line with the sensor). The setup can be seen in the left of Figure 3.9. Eighteen subjects volunteered for the study and performed one trial run at each of the three speeds (0.7 m/s, 1.1 m/s, and 1.5 m/s).

The sensor's accuracy was evaluated by approximating the difference between the displacement measured by the sensor in the ap-direction (corresponding to its y-axis) and the changing distance

between the markers on the sensor and rear of the calf measured by the cameras. Problems with data synchronization between the different systems and a suboptimal marker alignment made a more rigorous evaluation within the allotted time unfeasible. An example of a “good” correspondence between sensor and cameras is shown in the right of Figure 3.11.

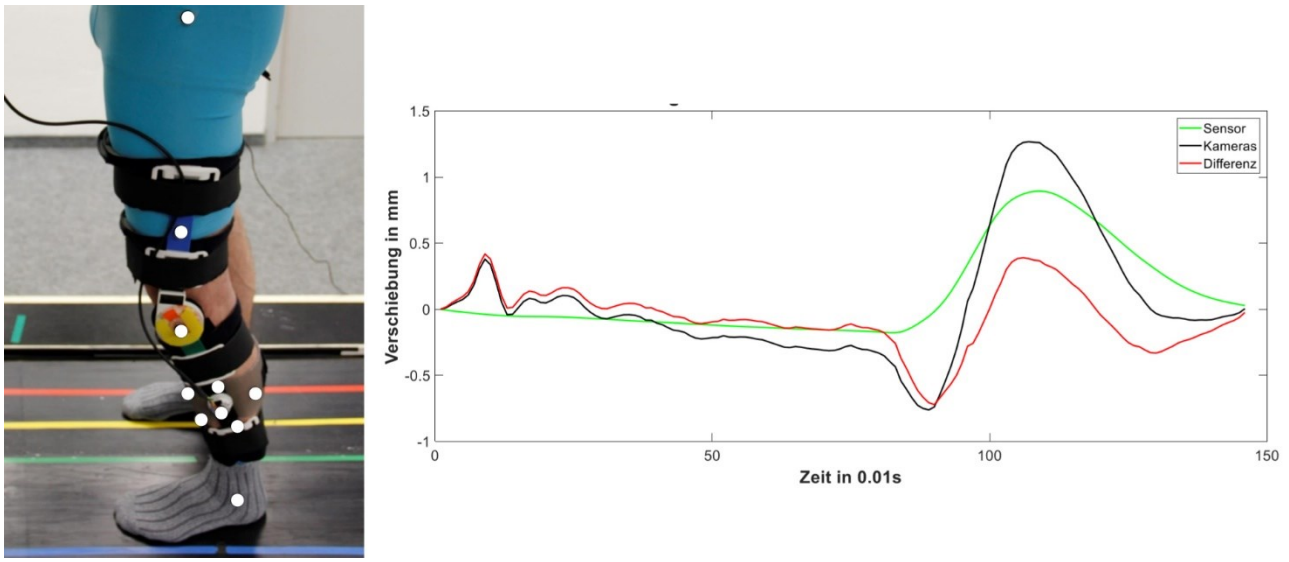

Figure 3.11: Subject wearing the orthosis, liner, and ten motion-capturing markers marked with white dots (left); example of a “good” agreement between sensor (green) and cameras (black), difference shown in red, for a 0.7 m/s walking speed. Difference at the peak is approximately 0.4 mm. Time in 0.01 s on the x-axis, displacement in mm on the y-axis.[29].

The trajectories of both displacements were similar overall but of different magnitudes, a fact that was attributed to the approximate nature of the analysis. The average difference between the two was increased with walking speed, which was partially attributed to the cameras being increasing imprecise at higher speeds. The knee angle was also calculated and the time of its peak compared with the time of maximum displacement. They were found to, by and large, occur at roughly the same time. The impact of distortion of the liner’s surface was not investigated.

3.5 Encountered Difficulties

Although their analyses were flawed[3], both Somogyi and the ARP showed that the system was performing significantly slower than expected: 15.5 Hz instead of 163.93 Hz (at 115 200 bps, 2000 fps, 40 bits over SPI, 536 bits over USB) [27] and either 64 Hz or 30 Hz instead of 452.26 Hz (at 115 200 bps, 12 000 fps, 40 bits over SPI, 136 bits over USB) [28]. As Somogyi was unable to increase the framerate above 2 000 fps and there was no evidence in the provided sketch that the ARP group

[3] It was assumed that SPI uses the asynchronous serial baud rate, and the bytes sent by the microcontroller to signal a register read were neglected. For data transfer between microcontroller and PC, it was assumed that the number of bytes sent corresponded to the size of the variable stored on the UNO when the data was being sent as strings with one byte per character sent; synchronization bits were also neglected.

solved this problem, it is doubtful if 12 000 fps was achieved. The higher of the two frequencies mentioned by the ARP worked well on one PC but resulted in numerous errors on another; the reduced frequency performed better. It was suggested that this was due to differences in PC hardware, but no information on the two systems was given. Even at the lower frequency, various errors still occurred at the beginning of measurements, primarily steps in time or seemingly random spikes in the other quantities. It was posited that these errors could be due to any number of a variety of sources: processing of the measurement hardware, the software, vibrations during measurement, the computer hardware, or the transmissions. It was noted that displacement data was sent with only two decimal places, which led to a loss of resolution and thus accuracy, particularly at peaks in the motion. Other areas of potential improvement were suggested by the ADP: a review of sensor settings, better software implementation for reading the data, and speed optimization of the software. The need for a better solution for data import was reiterated by Arnemann, who copied results from the ARDUINO IDE's serial monitor to EXCEL before further processing in MATLAB.

4 The Microcontroller

The ARDUINO UNO was chosen for the functional model out of convenience and adequately demonstrated the system's potential; it is, however, an entry-level model. As the functional model is to be developed into a measurement system with a sensor array for use in amputee studies, the capabilities of the system's microcontroller must be assessed and a new microcontroller with superior capabilities shall be selected. The performance of the new microcontroller will then be briefly compared to that of the UNO in preparation for software optimization in Chapter 5.

4.1 Requirements and Selection

The microcontroller's role is to read data from the sensor, process it to an appropriate degree, and then transmit the data to the PC. The five most important considerations are compatibility with the sensor, fast transmission speeds, fast processing speeds, the ability to accommodate multiple sensors, and the ability to react to a trigger signal. To be compatible with the sensor, the microcontroller must operate at either 3.3 V or 5 V and support SPI. Fast transmission and processing speeds are determined by the processor and the rated maximum SPI speed and baud rate. The ability to communicate with multiple sensors requires sufficiently many digital I/O pins to accept the addition SS lines. For the proposed system of four sensors, four SS pins are required; with an eye to possible future expansion of the system, the minimum requirement is increased to eight digital I/O pins. Finally, reacting to a trigger signal requires at least one analog input.

In addition to these mandatory requirements, there are two further specifications which might prove useful perks: large memory (EEPROM is currently being used to store the last calibration factors; SRAM and Flash are alternatives), and wireless connectivity. A generous amount of memory would make it possible to store calibration factors on the microcontroller (if this appears advantageous) or to achieve a wireless system by saving the data locally and uploading it to the PC at a later point in time. Built-in wireless connectivity, if the transmission speed is fast enough, is an obvious benefit; however, wireless connectivity can also be achieved with additional components. Cost, dimensions and weight should be within reasonable bounds. Immediate availability is also an important, although not crucial, consideration as it allows for a more rapid evaluation of potential.

To keep development time at a minimum by working off of the existing ARDUINO sketch, all the microcontrollers under consideration, with the exception of the MYRIO by NATIONAL INSTRUMENTS, are ARDUINO products. The MYRIO is included because it was the microcontroller originally recommended for the system. The other model under consideration at the time, the ARDUINO YUN, has since been discontinued. Table 4.1 summarizes each microcontroller's specifications. The most favorable ratings are colored green, acceptable ones yellow, sub-par ones orange, and poor ones red. The rating system is based upon comparisons between the microcontrollers.

Table 4.1: Microcontroller specifications summary from datasheets [55,56,60–63].

	UNO	MEGA	DUE	ZERO	MKR1000	MYRIO 1900
Mandatory						
Processor Speed (MHz)	16	16	84	48	48	667
SPI	yes	yes	yes	yes	yes	yes
Max SPI Speed (MHz)	8	8	42	8	8	4
Max Baud Rate (bps)	115 200	115 200	115 200	115 200	115 200	230 400
Analog Inputs	yes	yes	yes	yes	yes	yes
Digital I/O	14	54	54	20	8	32
Operating Voltage (V)	5	5	3.3	3.3	3.3	3.3 / 5
Perks						
EEPROM (kB)	1	4	no	no	no	no
SRAM	2 kB	8 kB	96 kB	32 kB	32 kB	256 MB
Flash	32 kB	256 kB	512 kB	256 kB	256 kB	512 MB
Wireless Connectivity	no	no	no	no	yes	yes
Dimensions (mm)	68.6 × 53.4	101.5 × 53.3	101.5 × 53.3	68.0 × 53.0	61.5 × 25.0	136.6 × 86.0
Weight (g)	54	37	36	12	32	N/A
Cost (€)	20	35	34-36	39	32	535+
Immediate Availability	yes	yes	yes	no	no	yes

All of the models meet the minimum requirements. The models differentiate themselves by processor and SPI frequencies and maximum baud rates. Among the ARDUINO boards, the DUE has the fastest processor (84 MHz) and the highest SPI frequency (42 MHz). The MYRIO has a significantly faster processor (667 MHz) and supports a higher baud rate (230 400 bps), but its SPI frequency is also the lowest of all the models considered (4 MHz). All models have a sufficient number of DI/O pins, although the MKR1000 provides only eight compared with the 32 of the MYRIO and the 54 of the MEGA and DUE. The MYRIO's storage capacity is far greater than any of the ARDUINO boards (256 MB SRAM, 512 MB Flash), of which the DUE has the greatest capacity (96 kB SRAM, 512 kB Flash); only the UNO and MEGA support EEPROM. Wireless connectivity is supported by both the MYRIO and MKR1000. The boards can be broadly categorized by their dimensions as small (UNO,

ZERO, MKR1000), medium (MEGA, DUE) and large (MYRIO). Weights for the Mega, DUE and MKR1000 range in the mid 30 grams, while the UNO is heavier (54 g) and the ZERO lighter (12 g); no information was available for the MYRIO, but it is likely the heaviest of all given its dimensions. All the ARDUINO boards are affordably priced at under 40€; in contrast, the MYRIO starts at 535€ for models with wireless capabilities. All but the ZERO and MKR1000 are immediately available at the institute.

In terms of performance, the two clear frontrunners are the MYRIO and DUE. The former boasts a faster processor and a higher baud rate, the latter a higher SPI frequency. A decision between them therefore determines which of these characteristics are more important to overall system performance. Reaching an informed decision would ordinarily require a more detailed analysis of the data quantities and processing times involved. However, a previous, unpublished attempt by a student to incorporate the MYRIO in the measurement system failed: it was unable to communicate with the sensor. The reason for the failure is unknown. Trusting that the student exhausted all reasonable avenues of inquiry and taking into account the additional development time of porting the sketch to LabView, the MYRIO is removed from contention.

The chosen microcontroller is the ARDUINO DUE, shown in Figure 4.1. It has six dedicated SPI pins, as indicated in the figure, and plentiful DI/O and analog in pins located around its perimeter (DI/O on the top and right-hand side, analog in on the bottom).

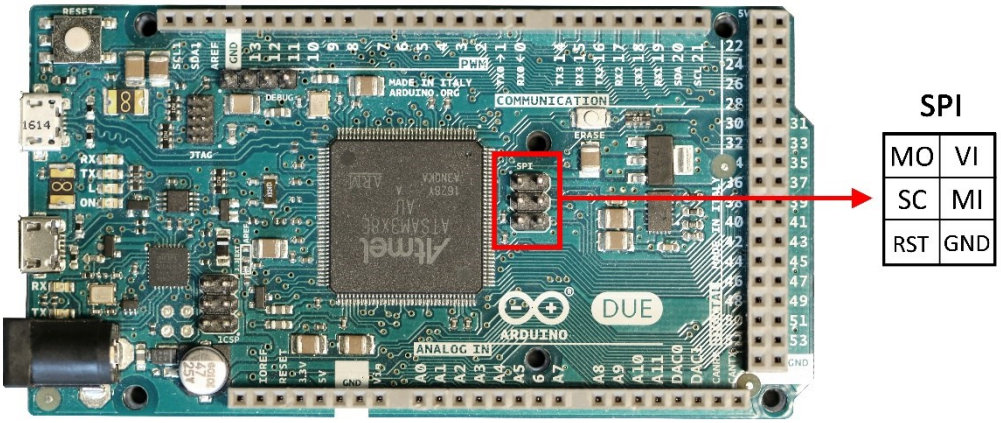

Figure 4.1: ARDUINO DUE and the location of its SPI pins.

4.2 Comparative Evaluation

Before improving the sketch's performance in Chapter 5, it is desirable to understand the starting point and quantify the improvements made solely through the use of a different microcontroller. To this end, the times required by the UNO and DUE, both for basic functions and the major functional blocks in the sketch will be assessed. This will also aid in preliminary identification of factors influencing performance.

4.2.1 Basic Functions

This section will consider the speed of a selection of basic functions integral to the sketch's `loop()`. Some are representative of a class of functions (e.g. an if statement or a calculation) and serve to gain a feeling for the times involved; others are simply commonly called functions that are essential to the sketch's functionality. The clock resolutions of the UNO and DUE are 4 µs and 1 µs respectively. The selected functions are described in more detail below:

`if (floor(micros()/5000)>1) {}`

A simplified if statement based off the sketch's check of whether a new frame has occurred; 5000 stands in for the frame period and 1 for the frame number.

`x=(2.252*25.4/8200)*2+1`

A simplified case of calculating the absolute displacement of X. Here, 2.252 is the calibration factor, 25.4 the conversion from inches to mm, 8200 the sensor resolution in counts per inch, 2 the "new" measured value in counts, and 1 the previous value in mm. All numbers are defined as 4-byte floats.

`millis()`

The time since the start of the program in milliseconds.

`micros()`

The time since the start of the program in microseconds.

`digitalWrite(pin, value)`

Writes the selected pin either HIGH or LOW.

Measurements are made by running the function of interest within a for loop for 2 500 repetitions. The time is taken in microseconds immediately before and immediately after the loop. An empty for loop was timed in an identical manner and its duration was found to be negligibly small on both microcontrollers. The difference between final and initial time is then determined and divided by the number of repetitions to arrive at the average time per operation (it must be noted that this process smooths out time fluctuations). The timing scheme is outlined here in Code 4.1.

Code 4.1: Timing scheme used within Arduino sketch.

```
N=2500;
T1=micros();
for (int i=0; i<N; i++) {
    >> insert function here <<
```

```
}
T2=micros();
T=(T2-T1)/N;
```

The results of the measurements are shown in Figure 4.2. Despite the UNO's temporal resolution being only 4 µs, averaging the measurement over a large number of repetitions makes it possible for resulting times per operation to below this threshold and thus a truer reflection of the actual times required.

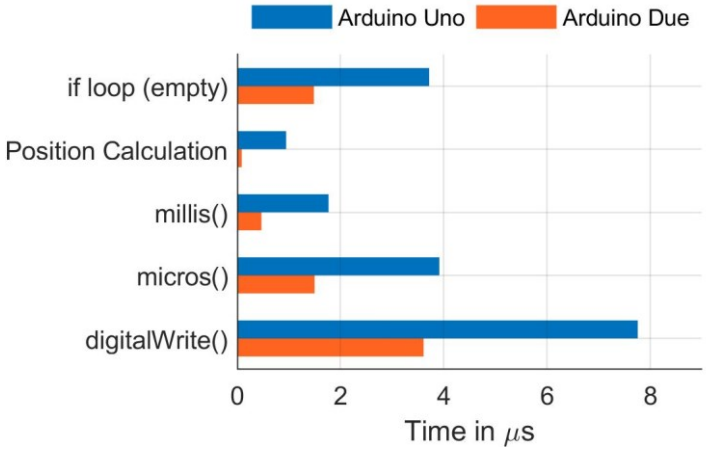

Figure 4.2: Average times per basic function.

The Due is significantly faster than the Uno for all measured functions. This is to be expected, considering the difference between their processor speeds. The results also give some insight into which functions may prove relevant to optimizing the speed of the sketch. At one end, it appears that calculating the absolute value (at least for the constant values employed here) takes a negligible amount of time; at the other, `digitalWrite()` is the most time intensive of the operations tested, requiring 3.61 µs on the DUE and 7.76 µs on the UNO. Time measurements in microseconds takes longer than those in milliseconds, suggesting that millisecond measurements should be preferred unless microseconds are required for precise timing. Finally, while not particularly expensive, it should be noted that the if loop is likely to take longer for more complex statements.

4.2.2 Functional Blocks

Broadly speaking, the original sketch can be divided into two blocks: reading and processing the sensor data (SPI Block), and sending the results to the PC (Serial Block). Without going into detail on the structure of these two parts, the two ARDUINO boards will be compared to establish a performance baseline. The SPI block is tested at the four fastest frequencies supported by the UNO: 8 MHz, 4 MHz, 2 MHz, and 1 MHz. Although the DUE is capable of frequencies up to 42 MHz, fre-

quencies above 8 MHz would not allow a direct performance comparison with the Uno and are not investigated here. The Serial Block uses the six standard baud rates supported by ARDUINO boards: 9 600 bps, 19 200 bps, 38 400 bps, 57 600 bps, 74 880 bps, and 115 200 bps. Each block is isolated and timed using the method presented in the previous section. For each setting, ten measurements of 1 000 cycles each were taken.

The results for the SPI Block are shown in Figure 4.3 and

Table 4.2. The behavior corresponds well to that which could be expected: an inverse dependence upon the clock frequency with the UNO being slower than the DUE due to the different processor speeds. Interestingly, the measurements reveal a difference in times depending on whether the sensor is in motion or stationary. The discrepancy is particularly striking on the UNO, where the stationary sensor case is approximately 65 µs faster. Although less pronounced on the DUE, there is still a difference of approximately 8 µs. In both cases, it remains approximately constant for all clock frequencies, suggesting that it is due to internal processes and not the SPI connection. While a stationary sensor results in standard deviations of 0.0 µs, a moving sensor results in elevated standard deviations, especially on the UNO. The behavior indicates that the discrepancy is related to processing carried out on the microcontrollers, but the exact cause is unclear at this stage and requires further investigation, as it may impact the consistency of system performance.

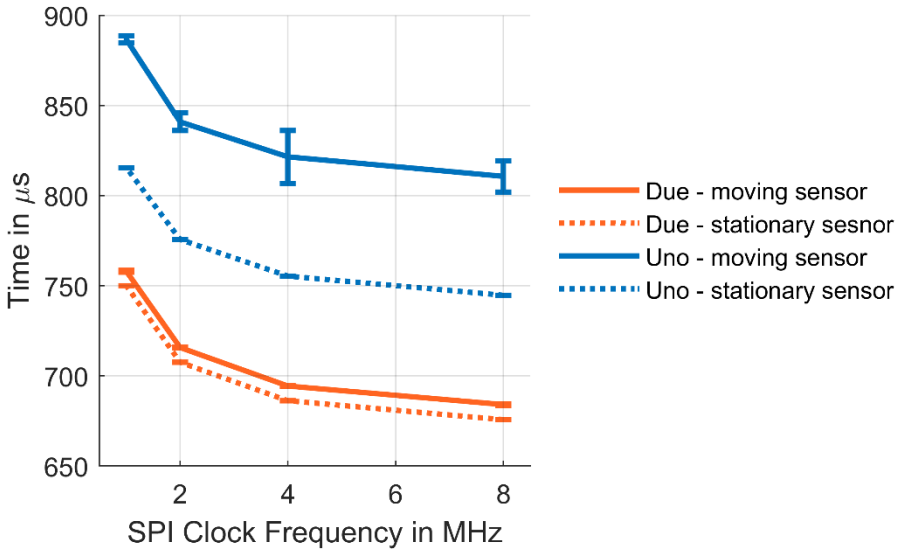

Figure 4.3: Times for the SPI block, means and standard deviations for ten trials consisting of 1 000 repetitions each.

Table 4.2: Times for the SPI block, means and standard deviations for ten trials consisting of 1 000 repetitions each.

Clock Frequency in MHz	Time in μs			
	UNO		DUE	
	stationary sensor	moving sensor	stationary sensor	moving sensor
8	744.6 ± 0.0	810.6 ± 8.6	675.8 ± 0.0	684.0 ± 0.4
4	755.4 ± 0.0	821.4 ± 14.7	686.3 ± 0.0	694.4 ± 0.2
2	775.5 ± 0.0	840.9 ± 4.9	707.5 ± 0.0	715.7 ± 1.2
1	815.4 ± 0.0	886.6 ± 1.9	750.0 ± 0.0	758.0 ± 0.6

The results for the Serial Block are given in Table 4.3. Since it is isolated from the SPI Block and therefore not receiving real sensor data, dummy data of the correct types (displacements as 0.00, SQUAL-value as 00, frame number as measuring loop number, and time as the current time in milliseconds). For each baud rate, ten trials of 1000 transmissions each were conducted; the mean time per data packet transmission is shown in Table 4.3. The output buffer was flushed after each transmission, so the presented times correspond to the times required for all of the data to leave the microcontroller. In practice, the times would be less as the sending function returns immediately and the data is sent using interrupts. The results show that the required times are approximately the same for both microcontrollers, which is to be expected given that most of the time is required for the transmissions themselves, not the returning of the function, and thus depend on the baud rate and not the processor speed. The lack of any standard deviation from the recorded measurements demonstrates the consistency of the transmission rates.

Table 4.3: Times for the Serial block, means times per transmission of dummy-data packet for ten trials of 1 000 transmissions each and flushing the output buffer after every transmission. Standard deviations were zero for all cases.

Baud Rate in bps	Time in μs	
	Uno	Due
9 600	53.856	53.937
19 200	26.868	26.904
38 400	13.375	13.338
57 600	8.705	8.881
74 880	6.889	6.802

115 200	4.339	4.370

One final point of interest is the maximum achievable frequency of the original sketch. SPI clock frequency was set to the UNO's maximum of 8 MHz and the baud rate to 115 200 bps. The sketch is run in its unaltered state. Throughout the measurements, the sensor is moved in a random fashion; the stationary case is not evaluated, as the sensor will be in motion when used in practice. Data is received in the ARDUINO IDE's serial monitor, from which it is copied into MATLAB. The timestamps are extracted and used to calculate the frequency. This approach eliminates any interference from MATLAB and thus reflects the maximum frequency that the ARDUINO sketch is capable of.

The calculated frequencies are 207.38 ± 18.35 Hz for the DUE and 211.66 ± 26.32 Hz for the UNO. The lack of meaningful difference between the two reflects the fact that data transmission, not internal processes, accounts for the majority of the required time. These values are significantly larger than those achieved in practice during previous work (15.5 Hz, 30 Hz and 64 Hz), supporting the theory that MATLAB is responsible for limiting the system's speed.

5 Speed Optimization

This chapter will first examine the original software in greater depth to understand how it functions and where there is improvement potential. The sketch and the GUI will then be analyzed and optimized individually. Finally, their combined performance will be evaluated.

Unless stated otherwise, the following PC hardware and software versions were used:

PC Hardware: Intel® Core™ i7-7700HQ CPU @ 2.80 GHz (3.80 GHz Turbo), 16 GB RAM

MATLAB: 7.14.0.739 R2012a

Arduino IDE: Arduino 1.8.5

5.1 Sensor Use

The sensor must be correctly powered up and have its firmware loaded prior to use. The necessary procedures are outlined in the datasheet [52] and will not be covered here as they are mandatory portions of the sketch and require no modification. Settings and data are stored in the sensor's registers, of which the relevant ones are:

- **Delta_X_L, Delta_X_H**: X movement counts since the last read; eight lower and upper bits of a 16-bit two's complement. Lower byte must be read first; each register clears upon reading. Range: –32768 to 32767. Read only.
- **Delta_Y_L, Delta_Y_H**: Y movement counts since last read; eight lower and upper bits of a 16-bit two's complement. Lower byte must be read first; each register clears upon reading. Range: –32768 to 32767. Read only.
- **SQUAL:** Surface quality register; upper eight bits of a 10-bit unsigned integer. Range: 0 to 169. The number of features seen equals four times the register value. Read only.
- **Configuration_I**: Sets the resolution on both X and Y or just the X axis. Read and write.
- **Frame_Period_Max_Bound_Lower, Frame_Period_Max_Bound_Upper**: Sets the maximum frame period (minimum frame rate) for automatic frame control or actual frame period (frame rate) for manual mode. Given in units of the internal oscillator clock, nominally 50 MHz. Read and write.
- **Frame_Period_Min_Bound_Lower, Frame_Period_Min_Bound_Upper**: Sets the minimum frame period (maximum frame rate) for automatic frame rate. 16-bit unsigned integer. Read and write.
- **Shutter_Max_Bound_Lower, Shutter_Max_Bound_Upper**: Sets the maximum allowable shutter value for automatic frame rate. 16-bit unsigned integer. Read and write.

As a manual frame rate will be used, the three values stored in the corresponding six registers must fulfill the following inequality:

Frame_Period_Max_Bound ≥ Frame_Period_Min_Bound + Shutter_Max_Bound

There is no protection for illegal values: if this inequality is not fulfilled, the sensor will not function. This was likely the case when the sensor appeared incapable of using framerates above 2 000 fps in previous work – if appears that only the Frame_Period_Max_Bound registers had been set, and the default values of the other registers did not satisfy the inequality.

A write operation is begun by the microcontroller driving SS low and sending a byte containing the register address with a "1" as the MSB. The microcontroller then sends a second byte containing the data. SS may be driven high after a delay of 20 µs. Subsequent commands may be issued after waiting an additional 120 µs.

A read operation is begun by the microcontroller driving SS low and sending a byte containing the register address with a "0" as the MSB. After a delay of 100 µs, the sensor sends the byte stored at the register address. SS may be driven high after a 120 ns delay. Subsequent commands may be issued after waiting a further 20 µs.

In the sketch, register addresses and values to be written are specified in hexadecimal (prefixed with `0x`).

5.2 Influence Factor Identification

Previous work encountered unexpectedly low frequencies and errors in the data. The causes of these problems were not further investigated, although numerous possible causes were mentioned. There are five areas of potential optimization:

1. Sensor settings
2. SPI communication
3. Arduino sketch
4. USB communication
5. MATLAB script and GUI

Naturally, these aspects cannot be considered in isolation as changes to one may affect another. To identify where the greatest improvement potentials lie, this section will examine the original sketch and GUI in detail before improvements are undertaken in Section 5.3.

5.2.1 Original Arduino Sketch

This section will present the overall structure of the sketch as well as in-depth analyses of the SPI and Serial blocks identified in Section 4.2.2. The relevant variables, their meanings and units are summarized in Table 5.1.

Table 5.1: Original sketch variables.

Variable	Meaning	Units	Format
`initComplete`	Confirmation of successful initialization; equal to 9 if true, 0 if false.	n/a	byte
`frame`	Frame number	n/a	long
`FS`	Frame period	µs	integer
`frequency`	Transmission frequency to PC	Hz	float
`dispTime_int`	Transmission period to PC (inverse of `frequency`)	ms	int
`dispTime_old`	Time of last transmission to PC	ms	unsigned long
`cpid`	Resolution	cpi	
`xydat`	Vector containing low and high bytes of X and Y	counts	byte
`xabs, yabs`	X and Y absolute displacement	mm	float
`xup, yup`	Maximum recorded displacement of X and Y	mm	float
`xlow, ylow`	Minimum recorded displacement of X and Y	mm	float
`x, y`	Pointer to the low bytes of X and Y in `xydat`	counts	byte
`kx, ky`	X and Y calibration factors	dimensionless	float
`squal`	Surface Quality value	n/a	byte

The discussion will focus on the `loop()` portion of the code shown in Code 5.1 and the functions used therein, which is responsible for its in-use performance. This is not to say that the `setup()` portion cannot be optimized as well, only that its impact on performance is negligible.

The `setup()` initializes serial and SPI communications using the specified settings. It then boots up the sensor as specified in the datasheet, checks six of its registers, and signals a successful initialization. During the sensor boot sequence and following the register check, status updates are sent to the PC.

The `loop()` consists of an if statement that checks whether a new image has been acquired by the sensor by comparing the current frame estimate (`micros()/FS`) to the previous frame number. If a new frame has been acquired, X and Y counts are read with `UpdatePointer()`, the SQUAL-value read with `readSqual()`, and the data sent to the PC by `printvalues()` at the specified frequency.

Code 5.1: Original sketch's `loop()` function.

```
void loop() {
     if (floor(micros()/FS)>frame) {
          frame++;
          UpdatePointer();
          readSqual();
          printvalues(frequency);
     }
}
```

5.2.1.1 The SPI Block

The SPI block consists of the `UpdatePointer()` and `readSqual()` functions (Code 5.2). Before executing, both check if the sensor was successfully initialized. This being the case, SS is driven low to begin SPI communication, the appropriate sensor registers are read with the `adns_read_reg()` function, and then SS is driven high again. `UpdatePointer()` then converts the lower bytes of X and Y from counts to millimeters and calculates the new absolute displacements. The new displacements are compared to the previous maximum and minimum values, which are updated if necessary.

Code 5.2: Original sketch's `UpdatePointer()` and `readSqual()` functions.

```
void UpdatePointer(void) {
     if (initComplete=9) {
          digitalWrite(SS,LOW);
          xydat[0]=(bytes)adns_read_reg(REG_Delta_X_L);
          xydat[1]=(byte)adns_read_reg(REG_Delta_X_H);
          xydat[2]=(byte)adns_read_reg(REG_Delta_Y_L);
          xydat[3]=(byte)adns_read_reg(REG_Delta_Y_H);
          digitalWrite(SS,HIGH);
          xabs=kx*(float(*x))/cpid*25.4)+xabs;
          yabs=ky*(float(*y))/cpid*25.4)+yabs;
          if (xabs>xup) {
               xup=xabs;
          } else if (xabs<xlow) {
               xlow=xabs;
          }
          if (yabs>yup) {
               yup=yabs;
          } else if (yabs<ylow) {
               ylow=yabs;
          }
```

```
     }
}
void readSqual() {
     if (initComplete=9) {
          digitalWrite(SS,LOW);
          squal=adns_read_reg(REG_SQUAL);
          digitalWrite(SS,HIGH);
     }
}
```

A theoretical analysis of these functions will now be presented using values based upon the measurements taken in Section 4.2.1 and theoretical SPI transmission times for an 8 MHz clock. When a value is prefixed with ≥, this indicates that the experimental value is considered a lower threshold. The derived theoretical times should be considered as minimums and will be compared with the experimental values.

The analysis begins with `adns_read_reg()` in Table 5.2. This is the function responsible for reading the sensor registers and it does so in strict accordance with the read process outlined in the datasheet and Section 5.1. The majority of the required 129.2 µs is due to the mandatory delays, 7.2 µs being due to the `digitalWrite()` calls to drive SS low and high.

Table 5.2: Theoretical breakdown of `adns_read_reg()`.

Operation	Duration in µs
`digitalWrite()`	3.6
`SPI.transfer(`1 byte`)`	1
`delay()`	100
`SPI.transfer(`1 byte`)`	1
`delay()`	1
`digitalWrite()`	3.6
`delay()`	19
`return()`	
Minimum:	129.2

The next function considered is `UpdatePointer()` (Table 5.3). After confirming that the sensor was started correctly, SS is driven low, each of the four displacement count registers are read using `adns_read_reg()`, and SS is driven high again. Finally, the absolute displacements are calculated and the new maximum and minimum values determined. The entire function is expected to require at least 528.7 µs.

Table 5.3: Theoretical breakdown of `UpdatePointer()`.

Operation	Duration in µs
`if (initComplete=9)`	≥ 1.5
`digitalWrite()`	3.6
4 × `adns_read_reg()`	516.8
`digitalWrite()`	3.6
2 × displacement calculation	≥ 0.2
2 × maximum/minimum	≥ 3
Minimum:	528.7

Lastly, `readSqual()` follows the same structure as `UpdatePointer()`, but without the additional processing (Table 5.4). It is expected to require at least 137.9 µs.

Table 5.4: Theoretical breakdown of `readSqual()`.

Operation	Duration in µs
`if (initComplete=9)`	≥ 1.5
`digitalWrite()`	3.6
`adns_read_reg()`	129.2
`digitalWrite()`	3.6
Minimum:	137.9

The total theoretical expected duration (the sum of the durations of `UpdatePointer()` and `readSqual()`) tallies to 666.6 µs. The experimental values are 675.8 µs for a stationary sensor and 684.0 μs for a moving one. These differ from the calculated value by 1.4% and 2.6% respectively.

While the correspondence between experiment and theory is good, the real value of this exercise is the inefficient structure of the code that it reveals. The sketch features numerous redundant calls to `digitalWrite()`. SS must be set low before SPI communication and high afterwards, but the number of transmissions permitted during this time is not limited to one, as the code assumes. If `UpdatePointer()` and `readSqual()` are combined, then only two calls are necessary – compared with the 14 currently used. The additional twelve calls waste 42 µs each loop. Additionally, checking that the sensor initiated properly each time one of these functions is called is superfluous; it should be checked once at the end of `setup()`. Finally, altering or removing the data processing

section (the calculation of absolute values and determination of new maxima and minima) will also save time.

5.2.1.2 Serial Block

The Serial Block consists of the `printvalues()` function (Code 5.3) which sends data to the PC at the specified frequency. It first determines if the elapsed time since last transmission is greater than the period corresponding to the transmission frequency. If this is the case, the time of last transmission is updated and the maximum, absolute and minimum X and Y displacements, current time in ms, frame number, and SQUAL-value are sent to the PC.

Code 5.3: Original sketch's `printvalues()` function

```
void printvalues(float frequency) {
     dispTime_int=int(1000/frequency);
     if (millis()-dispTime_old>dispTime_int) {
          dispTime_old=millis();
          Serial.println(xlow);
          Serial.println(xabs);
          Serial.println(xup);
          Serial.println(ylow);
          Serial.println(yabs);
          Serial,println(yup);
          Serial.println(millis());
          Serial.println(frame);
          Serial.println(squal);
     }
}
```

Determining the performance of the Serial block cannot be done as precisely as for the SPI block because of the transmission format. Although all of the variables are defined to consist of a specific number of bytes (see Table 5.5), they are sent as strings. The number of bytes transmitted thus depend on the number of characters necessary to represent the numbers as ASCII text and its only relation to the given variable sizes is its possible range. The number of characters depends on the magnitude, sign, and the number of decimal places being sent for floats. The minus sign and decimal point each add a byte. As `Serial.println()` is used instead of `Serial.print()`, a line break is included after each variable at the cost of two bytes each.

A total of nine variables are sent. The six variables relating to X and Y displacements are sent with a fixed two decimal places for a minimum of four bytes each. By definition, the SQUAL-value ranges between 0 and 169 and thus requires one to three bytes. Time and frame are both monotonically increasing. Since the time is in relation to the beginning of the program and `setup()` must be completed first, it will be in the 3500 ms range before measurements begin and therefore require a min-

imum of four bytes. Frame, on the other hand, begins at 1 and therefore requires at least one byte. All told, a minimum of 48 bytes must be sent and this quantity increases as time passes and new frames are taken.

Table 5.5: Calculation of the minimum number of bytes being sent over serial.

Variable	Type (Size in bytes)	Min. Data Bytes	Min. Total Bytes
xlow, xabs, xup	float (4)	4 each	6 each
ylow, yabs, yup	float (4)	4 each	6 each
time	unsigned long (4)	4	6
frame	long (4)	1	3
SQUAL-value	byte (4)	1	3
	Minimum Bytes per Transmission		48

Each byte sent consists of eight data bits and two synchronization bits (a start bit and a stop bit) for a total of ten bits per byte. Theoretical transmission times for 48 bytes at the various baud rates are given in Table 5.6. These are contrasted with experimental values without and with serial flushing after the nine variables are sent.

When data is transmitted from an Arduino over the serial port, the sending function (in this case `Serial.println()`) returns almost immediately; the data itself is periodically sent using interrupts. This ensures that the program is not slowed down by waiting for all of the data to be sent. The `Serial.flush()` command forces the sketch to halt and wait for all of the outgoing data to be transmitted. Including this function when timing gives a better indication of the actual transmission times involved instead of how long it takes for the transmitting function itself to return (although here they are very close because there are no other elements present in the code).

Comparing the measured times with the theoretical cannot give precise results, as the measurement conditions were such that the number of bytes increased beyond the minimum 48. It does, however, show that this estimate is not too far off the mark for the first moments of measurement.

Table 5.6: Theoretical and experimental serial transmission times in ms.

Baud Rate in bps	9600	19200	38400	57600	74880	115200
Theoretical (48 bytes)	50.00	25.00	12.50	8.33	6.41	4.16
Experimental – no serial flush	52.70	26.24	12.96	8.62	6.63	4.25
Experimental – serial flush	52.82	26.30	13.01	8.68	6.67	4.30

What is obvious from this analysis is that the “unstable” nature of sending data as strings is problematic. If a measurement is performed over a sufficiently long time or if it records sufficiently large displacements, the amount of transmitted data increases and fluctuates, which leads to fluctuating transmission times. Depending upon the desired transmission frequency, it is conceivable that this could slow the system down to an unacceptable degree. Further, as was shown in the earlier evaluation of serial transmission times (Section 4.2.2), sending the serial data is by far the most time intensive part of the entire process and a prime candidate for improvement. Possibilities include a change in the type of transmission (e.g. no line breaks or changing from strings to binary) and a reduction in the number of values sent.

5.2.2 Original MATLAB GUI

Variables relevant to this discussion are listed in Table 5.7. Some variables will not appear in subsequent code snippets and are presented for completion sake; others are present in the code snippets but are not relevant and thus not formally presented.

Table 5.7: Selected original GUI variables.

Variable	Meaning	Units
`s`	Serial port object	
`button1`	“Begin/End” toggle button – starts and ends an evaluation interval	
`button_state`	“Start/Stop” toggle button – starts and ends the measurement	
`a`	Dummy variable to store read data before further processing	
`xa, x ya, y`	Current and all values for absolute X and Y displacement (single, vector)	mm
`xlow0, xmin ylow0, ymin`	Current and all values for minimum X and Y displacement (single, vector)	mm

`xup0, xmax` `yup0, ymax`	Current and all values for maximum X and Y displacement (single, vector)	mm
`time, t`	Current and all times	ms
`frame, f`	Current and all frame numbers	
`squalv, squal`	Current and all SQUAL-values	

A measurement is begun by pressing the “Start/Stop” toggle button. Doing so directs the program to define the various variables used during data acquisition and configure the serial port. Once preparations are complete, the script enters a while loop that remains active until the button is pressed again to end the measurement. The basic structure is shown in Code 5.4. Bolded lines inside curly brackets indicate a section of code that will be presented in greater detail shortly. The script waits for 27 reads to be completed before commencing data acquisition; this is done to ignore the incoming confirmations and sensor registers sent during the DUE’s start up process. Once data acquisition commences, the variables are read in order, printed to the Command Window, and a completion message displayed in the Command Window. Finally, the relative error over the specified interval is calculated if the “Begin/End” button has been pressed a second time.

Code 5.4: Original GUI structure.

```
while button_state==1
     if (i>=27)
          {X MINIMUM}
          {X ABSOLUTE}
          {X MAXIMUM}
          {Y MINIMUM}
          {Y ABSOLUTE}
          {Y MAXIMUM}
          {TIME}
          {FRAME}
          {SQUAL}
          disp('ok');
          disp('--------------------------------');
     else
          fscanf(s,'%s');
     end
     {INTERVAL CALCULATION}
     i=i+1;
     pause(0.001)
     button_state=get(hObject,'Value');
     button1=get(handles.beginend,'Value');
     pause(0.001)
```

```
end
```

The **{X ABSOLUTE}** and **{Y ABSOLUTE}** sections are identical in structure and are shown by example of X in Code 5.5. The incoming string is read and converted to a scalar. It is then checked if an interval has been begun or ended during this iteration of the while loop. If so, the absolute value is saved to the appropriate variable containing either interval starts or ends. Finally, the newly read value (`xa`) is appended to the vector of previously received values (`x`).

Code 5.5: Original GUI - absolute displacement reading format.

```
a=fscanf(s,'%s')
xa=str2num(a);
if button1==1 & bx==1
     set(handles.x0,;'string',xa);
     bx=0;
     x0=[x0 xa];
end
if button1==0 & bx==0
     set(handles.xl,'string',xa);
     bx=1;
     x1=[x1 xa];
end
x=[x xa];
```

The **{X MINIMUM}**, **{X MAXIMUM}**, **{Y MINIMUM}**, and **{Y MAXIMUM}** sections are also identical in structure and are shown by example of X minimum in Code 5.6. The incoming strings are read and converted to a scalar. It is then checked if the new value is greater of less than the previously recorded global maximum or minimum; if so, the corresponding variable is updated (nothing further is done with these values). Finally, the newly read value (`xlow0` in this case) is appended to the vector of previously received values (`xmin`).

Code 5.6: Original GUI - minimum/maximum displacement reading format.

```
a=fscanf(s,'%s')
xlow0=str2num(a);
if (xlow0<xl)
     xl=xlow0;
end
xmin=[xmin xlow0];
```

The **{TIME}**, **{FRAME}**, and **{SQUAL}** sections once again share the same structure and are shown by example of time in Code 5.7. These are the simplest portions of code: the new string is read, converted to a scalar, and appended to the appropriate vector of previously received values.

Code 5.7: Original GUI - time, frame, and SQUAL-value reading format.

```
a=fscanf(s,'%s')
time=str2num(a);
t=[t time];
```

The **{INTERVAL CALCULATION}** will not be examined in greater detail as it is obsolete now that the GUI is no longer intended for determining the relative error over a given interval. To simplify the task of timing the GUI, this section was lumped together with section of code immediately below it containing the pauses and button updates.

Each of these functions was timed by bracketing it with MATLAB's stopwatch timer (`tic` and `toc`) with each new time appended to a vector of the previous ones. Dummy data (constants for X, Y, and SQUAL-values, ascending frame number, and actual time) was sent at 115 200 bps and the maximum transmission frequency from the DUE. Three measurements over approximately 300 s were taken for a total of 26 000-27 000 values per code section, which were then averaged. The results are shown in Table 5.8.

Table 5.8: Mean and standard deviation of times required of specified code sections; 115200 bps and maximum transmission frequency of dummy data from the Arduino, three repetitions taken over approximately 300 s for a total of 26 000-27 000 values each. Times given for a single execution of the section of code, i.e. the time for "absolute displacement" is for X or Y only, not both.

Script Section	Time in ms
Absolute Displacement	0.52 ± 1.00
Minimum or Maximum Displacement	2.88 ± 4.90
Time, Frame, or SQUAL-value	0.36 ± 0.78
Interval Calculation + Remainder	28.78 ± 5.53
Total Loop	33.91 ± 7.31

One striking result is that all code sections have relatively large variations; in the case of the portions in which data is read, the standard deviation is larger than the magnitude of the mean. Since the mean time cannot be negative, this indicates a concentration at the lower end of possible times with a spread of longer times above that. The durations of each loop for the first 4 000 values of a measurement are shown in Figure 5.1. Although part of this behavior may be attributable to the

various if statements, the actions contained within them should not be being executed as the related values are supposed to be constants.

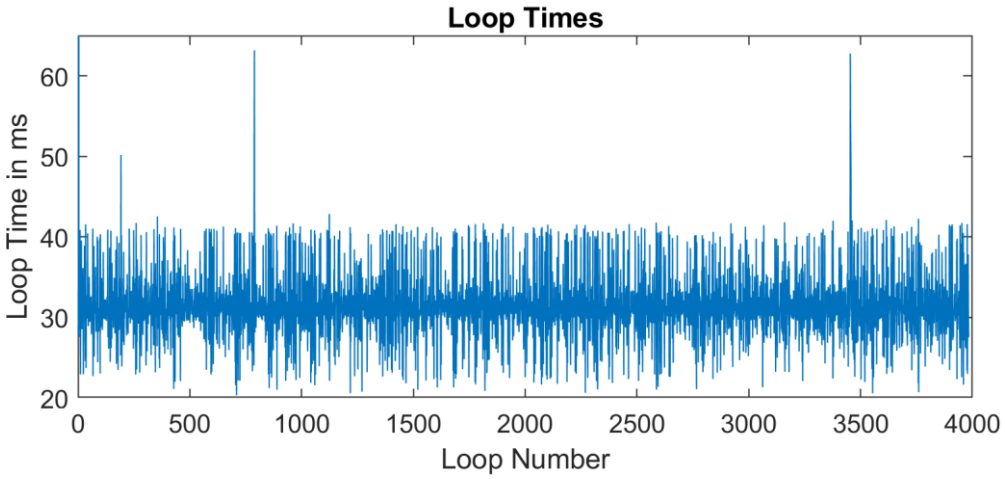

Figure 5.1: Original GUI loop times for dummy data send at maximum frequency from the DUE at 115 200 bps.

To get a better sense of what is truly going on, the values received for absolute X displacement over the same time period are plotted in Figure 5.2. The received values show regular fluctuations between the expected value and the values assigned to other variables; as they are constantly changing, the if-statements are indeed being activiated. Because the time, measured in milliseconds, dwarfs the other values, the finer detail can only be seen by zooming in on the smaller values ((b) in the figure). It appears that MATLAB is dropping values, suggesting that the input buffer is regularly overflowing.

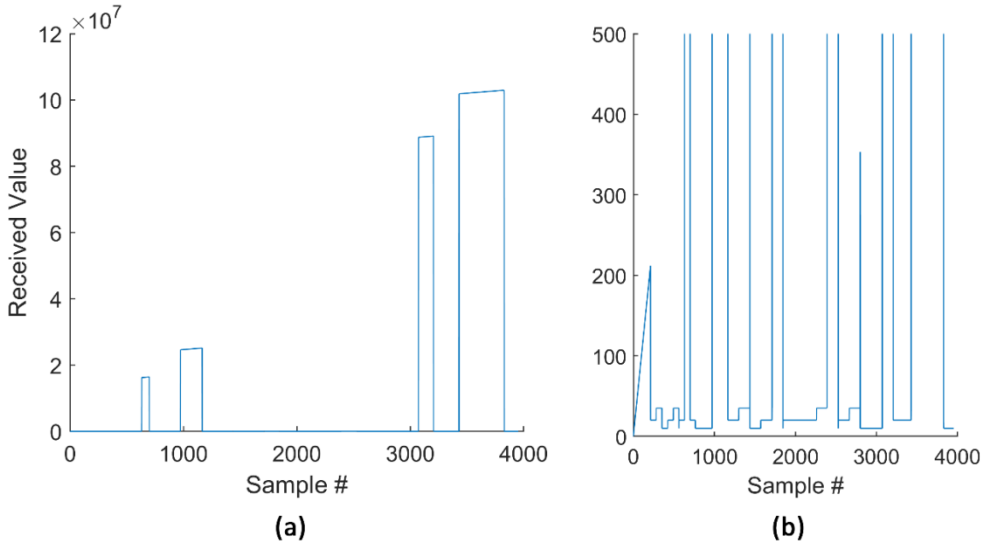

Figure 5.2: Original GUI, values received for absolute X displacement: full view (a) and small-scale detail (b).

To confirm this, another measurement is performed during which the number of bytes in the input buffer is recorded each cycle. The result is plotted in Figure 5.3 and confirms the suspicion: MATLAB is unable to read the incoming data fast enough and, very shortly after the measurement is begun, the serial buffer has reached its limit of 512 bytes. After this, the buffer repeatedly overflows, resulting in lost data, as reflected in Figure 5.2. This also accounts for the large variation in script section times – received values per variable are not constant and thus the various if statements are periodically activated.

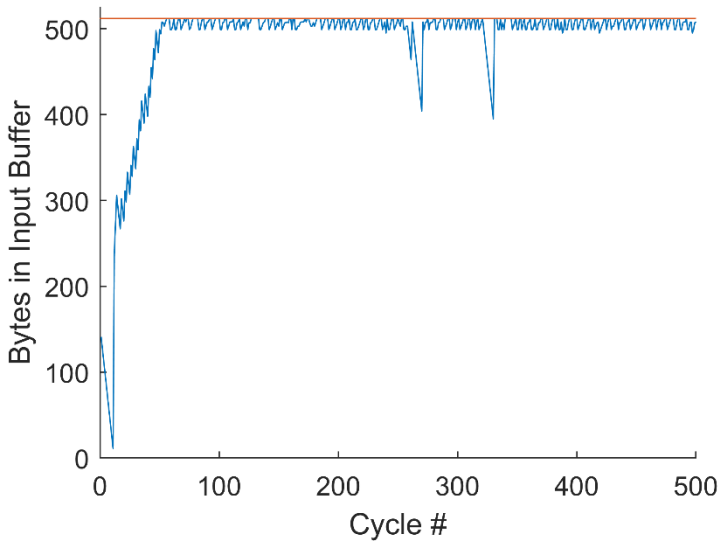

Figure 5.3: Original GUI, number of bytes in the serial buffer for transmission of dummy strings at 115 200 bps. Buffer limit of 512 kB indicate in orange.

One further observation from Table 5.8 is that most of the loop time is taken up by the interval calculation and remaining sections. Since interval calculation is never activated, something in the remainder must be inflating the time. Further investigation is beyond the scope of this assessment and will be carried out in Section 5.4.1. The maximum frequency achievable by the original GUI is 31.81 Hz. This is in accordance with the limit found by the ARP group; they did not encounter the overflowing buffer because they set the transmission frequency to be 30 Hz, which is within the GUI's capability.

5.2.3 Conclusions

A number of weaknesses in the current software have been revealed. The two greatest bottlenecks appear to be the data transmissions. Although the amount of data sent over SPI cannot be reduced, the clock frequency can be increased. The opposite is true of the USB transmission, where the speed is already set at the rated maximum, but an excessive number of variables are transmitted. This extra data not only takes longer to transmit, but it, along with some unnecessary functions, also bloat both the sketch and the GUI. In its current form, the system's limiting factor is MATLAB, which requires many times longer to complete one read cycle than expected. In the next sections, reading

of sensor data, transmission to the PC, and the internal workings of both sketch and GUI will be improved.

5.3 Improvements to the Sketch

Now that the original sketch structure has been evaluated, it is possible to make targeted improvements. These will begin with the manner in which the sensor registers are read and the SPI clock frequency. Following this, various functional and structural modifications will be undertaken. Lastly, the serial transmission method will be considered.

5.3.1 Register Read Mode

In the original sketch, the five sensor registers of interest are read individually and with great inefficiency (Section 5.2.1). This process may be greatly improved by consolidating its constituent parts. Instead of redundant `digitalWrite()` calls, SS is set low only at the beginning of the read and set high only once the read is complete. While SPI communication is open, all five registers are read sequentially and with the requisite delays between read operations. This new structure and its theoretical duration is shown in Table 5.9.

Table 5.9: Theoretical breakdown of the minimal duration required for optimized individual reading of registers.

	Operation	Time in µs
SS Low	`digitalWrite()`	3.6
Repeat 5 times	`SPI.transfer(`1 byte`)`	1
	`delay()`	100
	`SPI.transfer(`1 byte`)`	1
	`delay()`	20
SS High	`digitalWrite()`	3.6
	Total:	617.2

Since most of the time is taken up by mandatory delays, the new total of 617.2 µs is only a modest improvement over the original 666.6 µs. However, the ADNS-9800 may also be operated in burst mode. Burst mode enables a more rapid reading of the registers by continuously clocking data from them without specifying the register address or requiring the normal delays. It is activated by writing one byte to the Motion_Burst register. After a delay of one frame period, up to 14 registers may be read in fixed order without any further delays. The available registers, in order of reading, are:

BYTE [00] = Motion	BYTE [07] = Pixel_Sum
BYTE [01] = Observation	BYTE [08] = Maximum_Pixel
BYTE [02] = Delta_X_L	BYTE [09] = Minimum_Pixel
Byte [03] = Delta_X_H	BYTE [10] = Shutter_Upper
Byte [04] = Delta_Y_L	BYTE [11] = Shutter_Lower
Byte [05] = Delta_Y_H	BYTE [12] = Frame_Period_Lower
Byte [06] = SQUAL	Byte [13] = Frame_Period_Upper

Only the registers in the first column, specifically bytes 03-07, are of interest. However, while burst mode can be terminated at any time, the bytes must be read in the given order. This means that the first two bytes must be read before it is possible to read the X, Y, and SQUAL-value bytes, for a total of seven. Assuming an 8 MHz SPI clock frequency, each byte needs 1 µs for transmission. The true power of burst mode is derived from a single delay equal to the length of a frame period instead of a fixed 120 µs delay per read as in the individual case. For the minimum frame rate of 2 000 fps, the delay is 500 µs; for the maximum frame rate of 12 000 fps, the delay drops to 83.3 µs. Therefore, burst mode decreases the time required to read the sensor registers from the original 666.6 µs to 98.0 µs at 12 000 fps. The calculation is shown in Table 5.10.

Table 5.10: Theoretical breakdown of the minimal duration required for burst mode for framerates 2 000 fps and 12 000 fps.

		Time in µs	
	Operation	**2 000 fps**	**12 000 fps**
SS Low	`digitalWrite()`	3.6	3.6
	`delay(`1 frame period`)`	500	83.3
motion	`SPI.transfer(`1 byte`)`	1	1
obs.	`SPI.transfer(`1 byte`)`	1	1
x lower	`SPI.transfer(`1 byte`)`	1	1
x upper	`SPI.transfer(`1 byte`)`	1	1
y lower	`SPI.transfer(`1 byte`)`	1	1
y upper	`SPI.transfer(`1 byte`)`	1	1
SQUAL-value	`SPI.transfer(`1 byte`)`	1	1
SS High	`digitalWrite()`	3.6	3.6
	Total:	514.2	98.0

The impact of these changes upon the overall time required by the SPI block is measured experimentally. The duration of the altered `UpdatePointer()` function (having been combined with `readSqual()`) is measured in the same way as in previous sections. It is to be expected that the

experimental values are higher than the theoretical ones because the functions have not been optimized in any other way – the if-statement, calculation of absolute values, and determination of new maximum and minimum values are all still present. Taking the times in excess of the theoretical values less `digitalWrite()` as an estimate of the overhead yields 64.5 µs for minimized individual read, 33.5 µs for burst mode at 2 000 fps, and 33.2 µs for burst mode at 12 000 fps (all with an 8 MHz clock and stationary sensor). Switching to burst mode eliminates almost half of the overhead. Another point of interest is that the durations become more consistent for burst mode at 12 000 fps: its standard deviations are 1-2 µs compared with 3-5 µs for the other options. The differences are visualized in Figure 5.4 and summarized in Table 5.11.

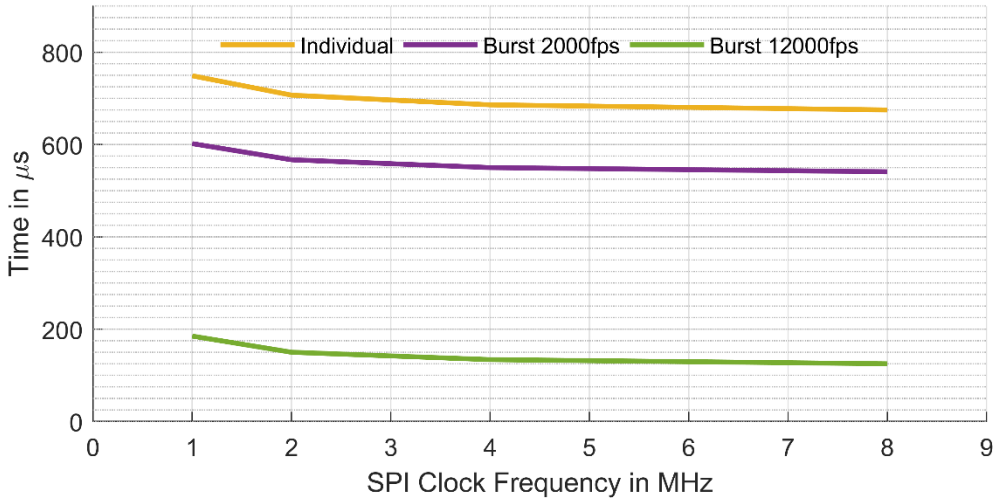

Figure 5.4: Times for reading the sensor registers using minimized individual reads (2 000 fps) and burst mode (2 000 fps and 12 000 fps). For clarity, only the results for a stationary sensor are shown and standard deviations are omitted because of their small scale relative to the figure window.

Table 5.11: Experimental times for different improved register read modes.

		SPI Clock Frequency			
Mode	**Sensor**	**1 MHz**	**2 MHz**	**4 MHz**	**8 MHz**
Original	stationary	749 ± 4	707 ± 4	686 ± 4	675 ± 4
(2 000 fps)	moving	758 ± 5	715 ± 5	695 ± 5	684 ± 5
Burst Mode	stationary	602 ± 3	567 ± 4	550 ± 4	545 ± 4
(2 000 fps)	moving	610 ± 4	576 ± 4	559 ± 4	551 ± 4
Burst Mode	stationary	185 ± 1	260 ± 1	134 ± 1	125 ± 1
(12 000 fps)	moving	192 ± 2	157 ± 2	140 ± 1	132 ± 1

One final area of optimization remains for reading the sensor registers, namely the SPI clock frequency. Until this point, it has been constrained to the four highest frequencies supported by all ARDUINO boards (8 MHz, 4 MHz, 2 MHz, and 1 MHz). One of the attractions of the ARDUINO DUE is that it supports frequencies up to 42 MHz. Measurements using all available clock frequencies be-

tween 4 MHz and 21 MHz are taken for burst mode at 12 000 fps. Frequencies greater than 21 MHz proved unstable and caused errors in the data. Although 21 MHz itself appears safe, it is nevertheless advisable to choose a slightly lower frequency that will be guaranteed to be reliable, especially since the difference between the next few options is insignificant. The results are shown in Figure 5.5. Some frequencies resulted in the same duration of the function. The standard deviations were consistently 1 µs for a stationary sensor and 3 µs for a moving sensor. The latter value is worse than those previously recorded; no clear reason for this discrepancy was identified, although it could be due to inconsistencies in the nature of the motion performed by the sensor.

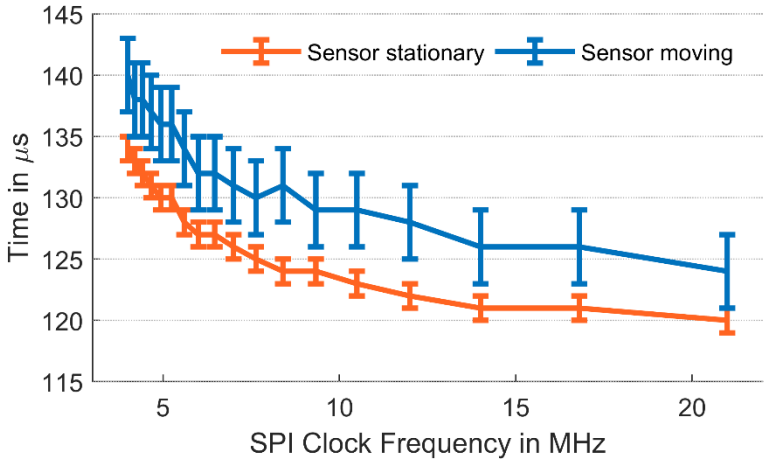

Figure 5.5: Burst mode times with various SPI clock frequencies for a stationary and moving sensor.

5.3.2 Structural and Functional Modifications

As was hinted at in the preceding section, the structure of the sketch can be optimized further, the aim being to reduce the remaining overhead as much as possible. To this end, various portions of the code are either removed completely or replaced with more efficient formulations.

Firstly, the relatively costly `digitalWrite()` is replaced by directly manipulating the ports with C commands [64]. This is done by using the following commands:

```
REG_PIOC_CODR=0x1<<29     // Set pin low
REG_PIOC_SODR=0x1<<29     // Set pin high
```

where **C** is the port name and **29** the pin number, corresponding to DI/O 10. Port and pin identifiers can be taken from (with appropriate caution) The Unofficial Arduino Due Pinout Diagram [65]. The difference between the C commands and `digitalWrite()` function are quantified by measuring the time required for 2 500 repetitions and calculating the average time per operation. Rounded

to one decimal place, the C commands required 0.3 µs compared with `digitalWrite()`'s 3.6 µs. Thus, the switch should save 6.6 µs per loop.

Next, superfluous sections of code are removed. The determination of maximum and minimum values will not be necessary for the final measurement system and can in any case be done once a measurement is completed. The if-statement checking if the sensor has been successfully initialized is also removed, as it is more appropriately located in `setup()`. Finally, since it is conceivable to simply relay the bytes read from the sensor registers without further processing, the effect of removing the calculation of absolute values is tested. The results for 4 MHz and 21 MHz SPI clock frequencies are shown in Table 5.12.

Table 5.12: Experimental times for making the outlined structural and functional changes with burst mode and a 12 000 fps frame rate; times in µs.

	SPI Clock Frequency			
	4 MHz		**21 MHz**	
Modifications	stationary sensor	moving sensor	stationary sensor	moving sensor
None	134 ± 1	140 ± 3	120 ± 1	124 ± 3
digitalWrite() → C write	129 ± 1	135 ± 3	115 ± 1	120 ± 3
Remove max/min	124 ± 1	130 ± 3	110 ± 1	115 ± 3
Remove if-statement	124 ± 1	130 ± 3	110 ± 1	115 ± 3
Remove calculation	113 ± 1	113 ± 1	99 ± 1	99 ± 1

The switch from `digitalWrite()` to C commands saves 4-5 µs, slightly less time than expected. Removing the determination of maximum and minimum values saved a further 5 µs. Although the if-statement turns out to have had a negligible impact on the time, its removal nonetheless cleans up the code. Through all these changes, the difference between times for a stationary or moving sensor remains constant at 5-6 µs; likewise, the difference in standard deviations between a stationary and a moving sensor remains constant at 2 µs. Removing the absolute value calculation not only saves 11 µs in the case of a stationary sensor, but it also eliminates any differences due to the sensor's motion. The final time for the 21 MHz clock is 99 ± 1 µs, which is 14.3% off from the theoretical value of 86.6 µs[4].

[4] Calculated as in Table 5.10, but with each SPI transmission requiring 0.38 µs (= 8 bits / 21 MHz).

5.3.3 Serial Transmission Methods

The final aspect of the sketch to be considered is the serial transmission method and amount of data to be sent. Previously, nine variables were sent to the PC; it is favorable to reduce this number. In accordance with the analysis of the preceding section, maximum and minimum values are omitted. The frame number is also omitted because it has no direct connection to the actual frame number captured by the sensor and is instead a counter of the number of loops completed. Thus, only four variables need to be sent: absolute X and Y displacements, SQUAL-value, and a timestamp. There are two possibilities for sending the data: either as strings (human-readable ASCII text), as in the original sketch, or as binary. The reading method and degree of post-processing required at the PC varies depending on which method is chosen.

The original sketch sends the data as strings using the `Serial.println()` command, which adds line breaks after each value. This is a particularly operator friendly method, as the user can easily read the incoming data. The line break does, however, add two bytes per variable. This can be avoided by using `Serial.print()` instead, but this has the disadvantage of lacking a terminator. This proves problematic when MATLAB reads the strings, as without a terminator, the read will never be completed. It is possible to specify a custom terminator character in MATLAB and thus reduce the extraneous bytes to one per variable. Both `Serial.println()` and `Serial.print()` suffer from the same weakness of increasing the number of bytes sent as time goes by and measured values change. The minimum number of bytes for each (using a custom one-character terminator for `Serial.print()`) are 17 and 21, for `Serial.print()` and `Serial.println()` respectively. The resulting minimum transmission times at 115200 bps are 1.48 ms and 1.82 ms.

Sending the data in binary is an attractive alternative, as the number of bytes per transmission is fixed by the variable type. With the absolute X and Y displacements defined as 4-byte floats, SQUAL-value as a byte, and the time as a 4-byte integer, a total of 13 bytes must be sent. At 115200 bps, this requires 1.13 ms.

Alternatively, it also possible to define the displacement as two-byte shorts, since it is reasonable to assume that the number of counts will not exceed its range (–32 768 to 32 767). In this case, a total of 9 bytes are sent, requiring 0.78 ms at 115 200 bps.

A summary of the number of bytes and transmission times by method is shown in Table 5.13.

Table 5.13: Total number of bytes sent for the different transmission methods and the times required at 115200 bps. A plus indicates a minimum byte number. Transmission times are theoretical and given in ms.

Variable	Type (Size)	`Serial.write()`	`Serial.write()`	`Serial.print()` + Terminator	`Serial.println()`
xabs	float (4)	2	4	5+	6+
yabs	float (4)	2	4	5+	6+
SQUAL-value	byte (1)	1	1	2 – 4	3 – 5
time[5]	integer (4)	4	4	5+	6+
Total Number of Bytes:		9	13	17+	21+
Transmission time at 115200bps in µs:		0.78	1.13	1.48+	1.82+

The downside to binary is that it is not human-readable and requires a more involved process to be sent from the Arduino and received by the PC. Binary transmissions use the `Serial.write()` command, which sends a single byte. The 2- or 4-byte variables must be decomposed into their component bytes and these then sent sequentially. One way of achieving this is with a union. A union allows different data types to be stored in the same memory location [66]. A sample union for the absolute X displacement is shown in Code 5.8. In this case, the union is given the union tag `X_val` and defined as consisting of the 4-byte integer[6] `count` and the four bytes of the array `data`. The variables may be accessed via the union variable `X` as `X.count` or `X.byte[i]` where `i` ∈ [0,1,2,3].

Code 5.8: Union for the absolute X displacement.

```
union X_val {
    int count;
    byte data[4];
} X;
```

Pending an analysis of the efficiency of data reception in MATLAB, it cannot be definitively stated which method will be chosen; however, the benefits of binary are obvious. The time cost is fixed and it thus promises predictable, stable transmission intervals. Even if reading and processing the

[5] Assumed to be beginning at approximately 3 500 ms, with times sent in ms.

[6] Note that count is defined as an integer, not a float. This is because it is referring to the integer number of counts read from the sensor register, not the float of the calculated absolute value. More on this in Section 5.5.

binary data in MATLAB takes longer than reading strings, it is unlikely that the former should take so much longer as to cancel out the time savings during transmission, especially over a longer period of time.

Finally, consideration must be given to the timing of the serial transmissions. In the original sketch, the condition for sending the next set of measurements was

```
if (millis()-dispTime_old>dispTime_int) {  >>send data<<  }
```

where `dispTime_old` is the time of last transmission and `dispTime_int` is the amount of time between transmissions, both in ms. The obvious problem is that this condition permits transmissions only *after* they are due. If this statement is checked even a millisecond before the intended transmission time, another iteration of the `loop()` will be performed first. This leads to a noticeably discrepancy between set and returned frequencies, often with sizeable steps between achievable frequencies.

The problem is solved by rephrasing the statement. Instead of checking if the elapsed time is greater than the transmission period, it checks if the elapsed time is within one sensor reading cycle[7] of the time when a transmission is due. If the elapsed time is within this window, then choosing to read the sensor one more time before sending the data would make the transmission late. Instead, the sketch waits until one microsecond before the due time and then begins the transmission, as shown in Code 5.9. Here, `time_prev` is the time of previous transmission, `time_disp` is the amount of time between transmissions, and `window` is the time required for a sensor read; all are now in microseconds for increased accuracy. This modification proves effective, and received periods are precise to within 14 µs at worst, with the average being 1.54 µs.

Code 5.9: Correction to achieve desired frequency.

```
if (micros()-time_prev>time_disp-window) {
              while (micros()-time_prev<time_disp-1) {}
                   >>send data<<
         }
```

[7] Using the experimentally measured value for `UpdatePointer()` using burst mode and at the appropriate SPI clock frequency.

5.4 Improvements to the GUI

The GUI's primary purpose is to ease data acquisition from the Arduino. As such, its core structure is straightforward and there are two possibilities for cutting time: removing unnecessary pieces of code and modifying the read and save methods.

5.4.1 Structural Modifications

The effect of modifying the GUI's structure is investigated here. The effect of changes is measured by bracketing the while loop in MATLAB's stopwatch function and recording the times for each cycle completed. Five trials of 30 s each were completed for each modification. Dummy data of the correct format was sent from the Arduino at the maximum possible frequency to ensure that the MATLAB script performs at its maximum capacity. Values given here differ and have smaller standard deviations than those presented in the initial analysis in Section 5.2.2 because the results here are averaged from the aggregate time of all cycles measured, not from each individual cycle's time. The issue of buffer overflow is disregarded as the interest here is in the cycle timing, not the integrity of the data. The results are summarized in Table 5.14.

The first modification is the removal of all code sections relating to calculating the relative error over a given interval as these are not be necessary in the final measurement system. The if-statement does serve the purpose of skipping the Arduino's startup transmissions, but this can be done in a more elegant fashion. It is removed to gauge its effect and will be replaced later. Next, all printing to the Command Window is removed (each read printed the received value, and each successfully completed cycle printed a two-line confirmation message).

The effect of these modifications is negligible – in fact, average time and standard deviation per cycle rose. Removing the if-statement could inflate the read times of the first 27 transmissions as these are longer than the rest, but should not have made such a large difference. In any case, the other changes should not increase the duration or its variation. No satisfactory explanation for this discrepancy was found.

The next area to investigate is the one identified as being the cause of the GUI's frequency cap. Remaining in this section are two `pause()` functions and one toggle button status update. Since removing the other status update (in the context of the interval calculation) had no measurable impact, it is unlikely that removing the second one will either. It is therefore reasonable to assume that the pauses are responsible for the delays. The effect of removing one of the two `pause()` functions almost halves the time per cycle from 34.92 ± 1.90 ms to 16.19 ± 0.14 ms; the maximum frequency correspondingly rose from 28.64 Hz to 61.77 Hz.

Table 5.14: Structural modifications to the GUI and resulting loop times (mean and standard deviations) and frequencies.

Modification	Time in ms	Maximum Frequency in Hz
None	32.41 ± 0.25	30.85
Remove interval calculation	33.07 ± 1.58	30.24
Remove if-statement	34.78 ± 2.56	28.75
Remove Command Window prints	34.92 ± 1.90	28.64
Remove one `pause()`	16.19 ± 0.14	61.77
Replace `pause(0.001)` with `drawnow()`	2.17 ± 0.03	460.83
Remove maximum, minimum, and frame reads	1.28 ± 0.01	781.28

Evidently, `pause()` is not behaving as expected. According to MATLAB's documentation, `pause(n)` pauses the program for `n` seconds where `n` can be a fraction. However, the actual resolution cannot be guaranteed and varies depending upon the scheduling resolution of the operating system and other system activity. For `n` with a 0.1 s resolution, the officially given relative error for MATLAB 2012 and 2018 is around 10%. Fractions of 0.001 s may be achievable, but the magnitude of the relative error will increase as the desired resolution becomes finer.

It should therefore not be expected that a pause of 0.001 s be achievable with any reasonable degree of accuracy – however, neither should it take 15 times longer. To confirm its duration and examine its behavior, the times taken by pauses of 0.001 s, 0.01 s, and 0.1 s are measured and averaged over 1000 repetitions. To investigate dependency upon MATLAB version, the measurements are repeated in MATLAB 2018[8]. The results are surprising. While MATLAB 2012 appears to be stuck at durations of 0.0156 s until pauses enter the 0.1 s range, no such problem exists in MATLAB 2018, which accurately reproduces all three pause durations (at least for the used PC hardware). If the ARP group used PCs with different version of MATLAB, it is possible that this was a contributing factor to their observation of different maximum frequencies between PCs.

Table 5.15: Times in required for pauses of various durations, all times in s.

	`pause(0.001)`	`pause(0.01)`	`pause(0.1)`
MATLAB 2012	0.0156	0.0156	0.1094
MATLAB 2018	0.001	0.0101	0.1001

Removing the second `pause()` improved performance by approximately the same amount. However, this result is not presented in Table 5.14 as it rendered the script incapable of registering chang-

[8] Version 9.4.0.813654 (R2018a); acquired only within the last month of thesis work.

es to button statuses – it was impossible to stop a measurement without manually aborting the entire script from the Command Window. A pause of some sort is integral to the code because it allows a moment of time for the GUI elements to update and register the user's input. However, given the subpar execution of `pause()`, leaving one in the code is an untenable solution. Fortunately, there exists a function `drawnow()` which *"flushes the event queue and updates the figure window"* [67]. Substituting `drawnow()` for the second `pause()` achieves the desired effect of avoiding the long delay while updating the GUI.

Finally, since they are no longer being sent, the section associated with reading and manipulating the maximum and minimum values is removed. This exhausts the possibilities of structural modifications to the GUI – all that remains within the while-loop is the reading and saving the four variables (absolute X and Y displacements, SQUAL-value, and time). The achieved cycle duration is 1.28 ± 0.01 ms.

5.4.2 Serial Reading Methods

Further improvements can be made to the serial read and save methods. This discussion will focus first on the case where data is sent as strings, then on the case where it is sent in binary. In the original GUI, the incoming strings were read with `fscanf()`, transformed into a scalar, and appended to a matrix in individual steps. It is conceivable that this is not the most efficient method.

The function `fscanf(obj,'format')` reads a string from the serial object `obj` (which is `s` in this case) and converts it to the specified format, e.g. `'%s'` for string or `'%f'` for float. These two formatting cases will be examined. If being formatted as a string, the value may be converted to a scalar. If converting, this may be done as a separate step, while saving to the storing vector, or while performing the read itself (this latter case is not investigated). Not converting to a scalar impacts the available saving methods: strings of various lengths cannot be saved to a matrix in a sensible way and require the use of a cell array instead.

A selection of possible combinations of read format, conversion, and save format were tested. Each combination was timed for ten trials of 1 000 reads of either one digit or ten digits. The results are tabulated in Appendix A1. No significant differences based upon the considered factors were demonstrated, and the manner of conversion and saving is thus a question of convenience.

Since the manner of reading, converting, and saving the incoming strings seems immaterial, the next thing to quantify is the time required to read strings of varying lengths. Measurements were performed in the same manner on incoming strings of lengths 1-10. The results are presented in Table 5.16.

Table 5.16: Means and standard deviations from ten repetitions of 1 000 reads of strings consisting of the indicated number of characters using `fscanf()` and their difference from the theoretical reception time of the corresponding number of bytes.

Number Characters	1	2	3	4	5
Time in µs	256.9 ± 7.5	342.7 ± 1.6	428.4 ± 1.5	514.4 ± 1.4	600.0 ± 2.4
Difference from theoretical time in µs	– 0.46	– 0.48	– 0.61	– 0.4	– 0.64
Number Characters	6	7	8	9	10
Time in µs	685.4 ± 3.5	771.6 ± 2.3	857.0 ± 2.8	943.0 ± 2.7	1029.0 ± 1.9
Difference from theoretical time in µs	– 1.01111	– 0.62	– 0.96	– 0.8	– 0.62

The average difference in time between neighboring digits is 85.8 µs, which differs by only 1.18% from the theoretical transmission time for one byte at 115 200 bps. Subtracting this value per byte received (one per character plus two for the line break) yields the difference from theoretical time. The results show that the serial read times approximately equal the time required for the respective number of bytes to arrive. The time required for `fscanf()` appears to be negligible by comparison and could not be definitively identified.

Binary data is read with the `fread(obj,size,'precision')` function where `obj` is again the serial port object `s`, `size` the number of values to be read, and `precision` the desired format (e.g. 32-bit integer). Binary transmissions read as 8-bit, 16-bit, and 32-bit integers were also timed and the results are shown in Table 5.17.

Table 5.17: Means and standard deviations from ten repetitions of 1000 reads of binary integers consisting of the indicated number of bytes using `fread()` and their difference from the theoretical reception time.

Number Bytes Read	1	2	4
Time in μs	157.8 ± 3.7	170.6 ± 11.7	342.9 ± 2.1
Difference from theoretical time in μs	72.0	– 0.4	– 0.3

Here, the large standard deviation for reading two bytes is unexpected and the difference between reading one byte and two bytes is 13.5 µs, which is clearly too small to be the time required for the second byte to arrive. However, that correspondence is restored for the difference between two and four bytes, which is 171.7 µs or 85.8 µs per byte. This is the same as the average difference observed for reading strings. Subtracting this value per byte received again shows that the time re-

quired to read the incoming bytes is approximately identical to the time required for them to arrive. The reason for the greater time required to read a single byte is unclear.

As the necessary read times correspond with the required transmission times per byte, the time per read for each transmission format is approximately equal to that predicted by theory in Table 5.13 in Section 5.3.3. Even for the minimum number of characters sent in the strings, the times derived from the values measured here for the two binary options are significantly faster (0.84 µs for the 9-byte version, 1.19 µs for the 13-byte version) than the two string options (1.46 µs with one terminator character and 1.80 µs with two). It is therefore determined that data shall be transmitted in the binary format.

5.4.3 MATLAB Limitations and REALTERM

The times given in the preceding section are the times required for data transmission only. The time required for an individual data reading and transmission cycle will be greater due to the time required for reading the sensor registers. The actual reception frequencies are investigated by running the GUI with the sketch and sensor. The timestamps sent from the DUE are used to determine the transmission frequency. Recalling that MATLAB's input buffer previously began to fill at higher frequencies, the number of bytes in the input buffer is measured each cycle; this has a negligible impact on system timing. The behavior of the buffer's capacity is used to determine if MATLAB can perform reliably at a given frequency. If the number of bytes rises steadily and approaches the buffer limit, the frequency is deemed unreliable. Frequencies are specified in 50 Hz increments between 50 Hz and 1000 Hz and measurements are performed for approximately ten seconds, which is long enough to get a good measure of the transmission frequency, but not so long that the buffer will fill and cause further processing to be required before calculating the frequency. Measurements are conducted for both strings and binary.

Transmission using strings with two terminator characters attains a maximum frequency of about 442 Hz. Transmissions using binary achieve maximum frequencies of 798 Hz for the 13-byte version and 1109 Hz for the 9-byte version. However, MATLAB is only able to read data reliably up to the 540-550 Hz range. After this, the buffer begins to fill and errors are guaranteed to occur when it overflows. For a measurement system with four sensors, this equates to a maximum frequency of 137.5 Hz per sensor. This is an acceptable value and a significant improvement over previous performance, but the system is clearly capable of even greater frequencies. MATLAB is the limiting factor.

It is of course possible to increase the size of the serial input buffer (no maximum value is specified by MATLAB, and such is therefore likely to depend on PC hardware), but this does not fix the problem, it only delays the inevitable. If the rate of incoming data exceeds the rate of reading, the buffer will fill if the measurement continues for long enough.

There is no requirement that MATLAB be used for data acquisition; it is merely a convenience. If a MATLAB GUI presents the most user-friendly means of acquiring and saving data, then a simple seri-

al port monitor presents the fastest. One such program is built into the ARDUINO IDE. Using it, settings can still be sent to the Due (albeit by hand), and the received data could then be copied from the terminal and saved as a text or binary file to be imported into MATLAB later. Arnemann used this the minimalist option.

There are other programs available with more features. SparkFun has assembled a list of recommended terminal programs, including the ARDUINO SERIAL MONITOR, WINDOWS HYPERTERMINAL, TERATERM, REALTERM, YAT, COOLTERM, and COMMAND LINE (for Windows) [68]; many more exist. The recommendations give an overview of key features of each program as well as a short list of pros and cons; the descriptions are by no means complete, but rather touch only what the author considers to be highlights. To be considered for the measurement system, the terminal program must support 115200 bps, be able to read binary, and send data in either binary or string format. The ability to save received data or interface with MATLAB are positives. Of the programs mentioned, only REALTERM offered compatibility with MATLAB while fulfilling all the other requirements. REALTERM is available free from SourceForge [69] and is intended, among other things, for dealing with large streams of binary data. A screenshot of REALTERM's interface is shown in Figure 5.6.

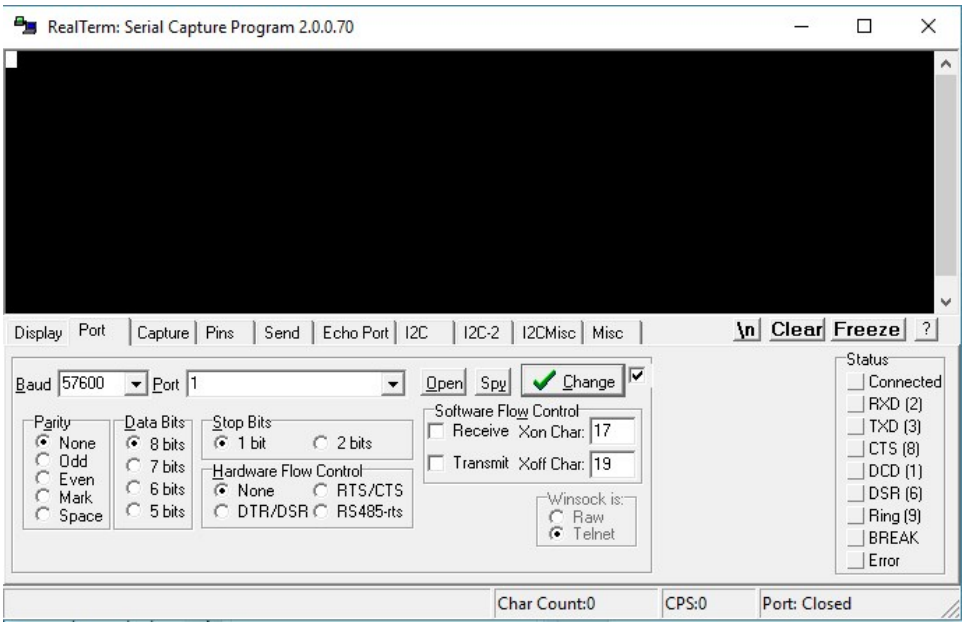

Figure 5.6: Screenshot of REALTERM open to the "Port" tab.

The ability to interface with MATLAB means that the user-friendly GUI can still be used to enter settings and control the measurement without the measurement being subject to the speed restrictions imposed by MATLAB. Implementation is not as straightforward as expected, however – documentation for this aspect of REALTERM is sparse. Provided examples are adapted [70] and some functions guessed at to achieve the desired result. The basic structure of the new code is presented in Code 5.10 below.

Code 5.10: REALTERM implementation in MATLAB

```
RT=actxserver('realterm.realtermintf');
RT.baud=115200;
RT.Port=com(4:end);
RT.PortOpen=str2num(com(4:end));
RT.CaptureFile=save_loc;
invoke(RT,'startcapture');
invoke(RT,'putchar','…');  /  invoke(RT,'putstring','…');

    { >> data acquisition << }

invoke(RT,'stopcapture');
invoke(RT,'close');
delete(RT);
```

The first step is to open REALTERM, configure the baud rate and COM port, and finally open the port. The file location for saving the binary data is set (`save_loc`) and data saving begun. MATLAB then sends the settings through REALTERM with either the `put_char` or `put_string` commands, depending on the size of the variable. Following this, the ARDUINO returns the customary status updates, sensor registers (if desired), and, if all has gone well, the data. Since MATLAB can open the binary file while data is being collected and saved to it, it can immediately read the status and register information and act accordingly (e.g. display the registers in the GUI or abort the measurement if an error is detected). However, reading the measurement data is left until after the process is complete. Once the measurement is ended, MATLAB stops the data saving process and closes REALTERM.

With RealTerm, frequencies of 899.5 Hz for the 13-byte version and 1298.7 Hz for the 9-byte version are achievable. The limitations posed by MATLAB have been effectively overcome. It must be noted that, while the interface between MATLAB and REALTERM (sending settings, saving and reading data), appears to work perfectly, there are problems when attempting to close REALTERM. The program does close, but always throws an error message in the background. Unfortunately, there does not appear to be any documentation of the meaning of the error codes. No adverse effects have been noted, but closing the error messages by hand rapidly grows tiresome when multiple measurements have been taken.

5.5 Summary and Expansion for Sensor Array

The sketch and GUI are reworked to reflect the changes made and insights gained in the preceding sections and then expanded to accommodate multiple sensors. The final sketches (one each for 9-byte and 13-byte versions) and the GUI are included in the Digital Appendix. System settings are configured as follows:

Table 5.18: System settings, maximum frequencies for the two versions rounded up.

Setting	Value	Units	Notes
Frame Rate	12000	fps	(manual[9])
Resolution	8200	cpi	
SPI Clock Frequency	14	MHz	(clock divider 6)
SPI Data Transmission	8	bytes	(using burst mode)
Baud Rate	115200	bps	
Serial Data Transmission	13	bytes	
Serial Data Transmission Frequency	900	Hz	(maximum)
Serial Data Transmission	9	bytes	
Serial Data Transmission Frequency	1299	Hz	(maximum)

The sketch has been modified to accept up to eight sensors by predefining pins to be used, converting the relevant variables to vectors or arrays, and looping over the number of sensors being used. Calibration and calculation of absolute displacements has been shifted offline, and the only intermediate processing carried out on the DUE is to convert the X and Y counts from the twos complement read from the registers.

Definition of the slave select pins is shown in Code 5.11, where `SS_pins` are the pin numbers on Port C corresponding to the physical DI/O pins specified in `SS_set`. They apparently redundant definition is because `SS_pins` are used during the data acquisition process, while `SS_set` are easier to use when configuring the pins as outputs during the setup (the built-in `pinMode()` function can be used directly).

[9] Inequality fulfilled by: Frame_Period_Max_Bound = 4167, Frame_Period_Min_Bound = 4000, and Shutter_Max_Bound = 150

Code 5.11: SS pin designations.

```
int SS_pins[8]={28,26,25,24,23,22,21,29};
int SS_set[8]={3,4,5,6,7,8,9,10};
```

The SQUAL-value is defined as a vector, while the X and Y counts and time are defined as arrays within their unions (exemplified here by X counts):

Code 5.12: Definitions of X displacement for 13-byte version and SQUAL-value.

```
union X_val {
    int count[8];
    byte data[8][4];
} X;

byte squal[8];
```

The `loop()` now cycles through sensor reading and data transmission by sensor number. The original `UpdatePointer()` and `readSqual()` functions have been combined in the new `UpdateData()` function, while `printvalues()` has been similarly reworked into `PrintValues()`. `UpdateData()` reads the necessary sensor registers using burst mode. The functions `com_begin` and `com_end` have been modified to set the SS pins low or high by directly manipulating the appropriate pins on Port C as shown in Section 5.3.2. Calibration and calculation of absolute displacements has been moved offline so that the only processing performed by the Due is to convert the twos complement register values for X and Y counts. The new sensor reading function is shown in Code 5.13, with `FS` being the frame period and `burst[]` a temporary storage vector for the sensor data.

Code 5.13: Revised SPI Block - `UpdatePointer()` and `readSqual()` combined into `UpdateData()`.

```
void UpdateData(int i) {
    com_begin(i);
    SPI.transfer(REG_Motion_Burst & 0x7f);
    delayMicroseconds(FS);
    burst[0]=SPI.transfer(0);
    burst[0]=SPI.transfer(0);
    burst[0]=SPI.transfer(0);
    burst[1]=SPI.transfer(0);
    burst[2]=SPI.transfer(0);
    burst[3]=SPI.transfer(0);
    squal[i]=SPI.transfer(0);
    com_end(i);
```

```
    delayMicroseconds(1);
    X.count[i]=convTwosComp(burst[0]);
    Y.count[i]=convTwosComp(burst[2]);
}
```

Data transmission timing has been improved as discussed in Section 5.3.3 and each sensor's data is transmitted in binary as shown in Code 5.14 for the 13-byte case.

Code 5.14: Revised Serial Block function `PrintValues()` for 13-byte version.

```
void PrintValues(int i) {
    if (micros()-T.time_prev[i]>time_disp-window[SS_num-1]) {
        while (micros()-T.time_prev[i]<time_disp-1) {}
        T.time_prev[i]=micros();
        Serial.write(X.data[i][0]);
        Serial.write(X.data[i][1]);
        Serial.write(X.data[i][2]);
        Serial.write(X.data[i][3]);
        Serial.write(Y.data[i][0]);
        Serial.write(Y.data[i][1]);
        Serial.write(Y.data[i][2]);
        Serial.write(Y.data[i][3]);
        Serial.write(squal[i]);
        Serial.write(T.data[i][0]);
        Serial.write(T.data[i][1]);
        Serial.write(T.data[i][2]);
        Serial.write(T.data[i][3]);
    }
}
```

The sketch has also been expanded to allow the GUI more control over the measurement settings and flow, while the GUI itself has also been greatly expanded to provide a more intuitive and comprehensive experience for the user. These features will be explained in greater detail in Chapter 7 and as needed in Chapter 6. For now, the discussion of the GUI will be restricted to data reception and pre-processing sections.

The portion of the GUI using REALTERM was previously shown in Section 5.4.3 and it was mentioned that MATLAB performs checks upon the start up confirmations sent by the Arduino. The importing, processing and saving of the data occurs after the measurements are complete. With the binary file generated by REALTERM opened as incoming, the data is first read into one long vector as shown in Code 5.15, where the first line uses previously determined factors to ensure that no in-

complete sets are transcribed and that there are thus an equal number of values for each variable of each sensor.

Code 5.15: Revised data reading portion of the GUI, shown for 13-byte version.

```
for k=1:sensors*(N-E-mod(N,13*sensors))/E
    R(end+1)=fread(incoming,1,'int32');
    R(end+1)=fread(incoming,1,'int32');
    R(end+1)=fread(incoming,1,'uint8');
    R(end+1)=fread(incoming,1,'int32');
end
```

The data is then scaled with the calibration factor and reshaped into a matrix organized by row (i.e. rows 1 – 4 are X counts, Y counts, SQUAL-value, and times of sensor 1, rows 5 – 8 belong to sensor 2, etc.). Following that, the absolute X and Y displacements are computed as shown in Code 5.16. The first line is again ensuring equal numbers of entries for each sensor and `factors` contain each sensor's X and Y calibration factors (ordered as: X1, Y1, X2, Y2, X3, Y3, …). SQUAL-value and the time are left unprocessed, although it is conceivable to remove the time's start bias. The script is not the most efficient means of solving the problem, but as at this point in the process time is non-essential, it serves adequately.

Code 5.16: Revised data processing portion of the GUI, shown for 13-byte version.

```
k=1;
for q=1:(4*sensors):(W(1)-mod(W(1),4*sensors))
    for p=1:4*sensors
        Datasort(p,k)=factors(p)*R(q+p-1);
    end
    k=k+1;
end
for i=1:4:4*sensors
    AbsValCalc(1:2,1)=0;
    for j=1:length(DataSort)-1
        AbsValCalc(1:2,j+1)=AbsValCalc(1:2,j)+(25.4/8200)
                                              *DataSort(i:i+1,j);
    end
    DataSort(i:i+1,:)=AbsValCalc(1:2,:);
    clear AbsValCalc;
end
```

The sketch and GUI, as described here (13-byte version), are used in the following Chapter.

6 Test Rig Evaluation of System Functionality

Now that the system's software has been optimized, it is time to test the improved performance in a controlled environment. The performance of the system using four sensors will be evaluated on a test rig. Experiments aim to determine the best calibration method as well as the system's overall accuracy and precision.

6.1 Introduction and Experimental Setup

Functionality of the sensor array is tested using a bi-axial test rig provided by the Institute of Production Management, Technology and Machine Tools at the Technische Universität Darmstadt. Two synchronous, digital AC servo motors[10] controlled by REXROTH INDRADRIVE CS each move a linear ball screw drive[11]. The linear drives are mounted perpendicular to each other and move a mounted cantilever in the X and Y directions over the base plate.

A square of liner material 5, the Medi Liner Relax, is placed on the base plate. It measures approximately 10×13 cm with a thickness of 2.- 3.5 mm, changing smoothly across one diagonal of the material. Figure 6.1 shows the liner's appearance against a dark background to the naked eye as well as the microstructure.

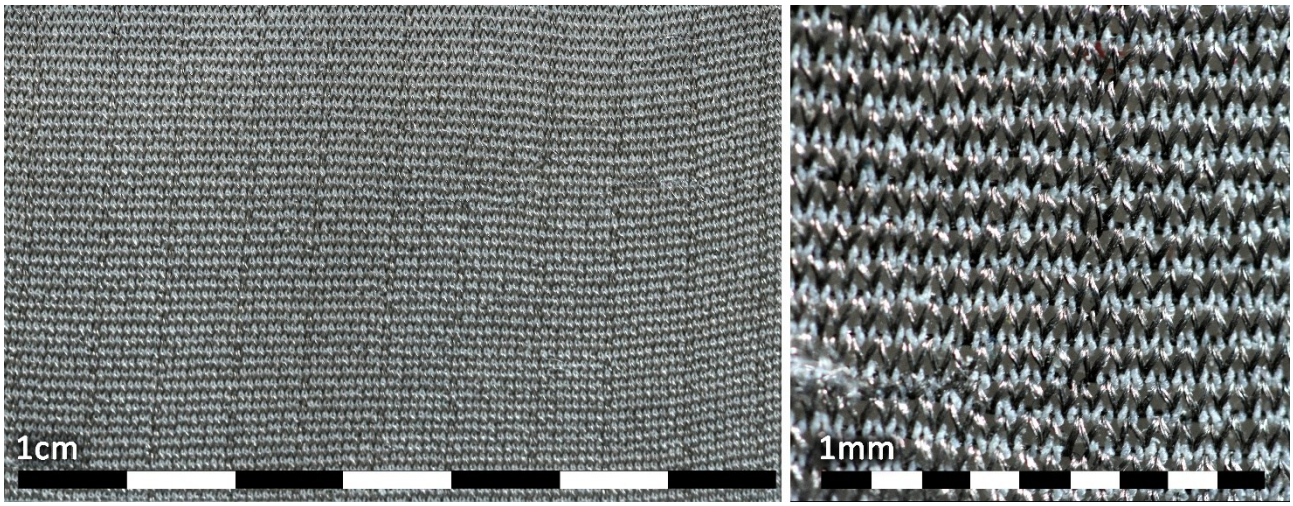

Figure 6.1: Medi Liner Relax, appearance to naked eye (left) and microstructure (right). Photographs taken with Canon EOS Rebel T6i and Canon 100mm macro lens at f/3.2, 1/100 sec, ISO 1600; contrast enhanced.

Four sensors are mounted in a 3D-printed bracket (Figure 6.2), which is screwed to the aluminum cantilever. The slots ensure that the height can be correctly set so that the base of the bracket is in contact with the liner. The base extends 2.4 mm below the lens to ensure a constant distance be-

[10] REXROTH CKK 9-70

[11] REXROTH MSK030B-0900-NN-M1-UG1-NNNN

tween sensor and liner. As the influence of upper and lower hole diameter had been previously investigated and was not a focus of this evaluation, the “hole” was simply a cutout in the shape of the entire lens. As larger holes were found to be better for accuracy, no adverse effects are expected.

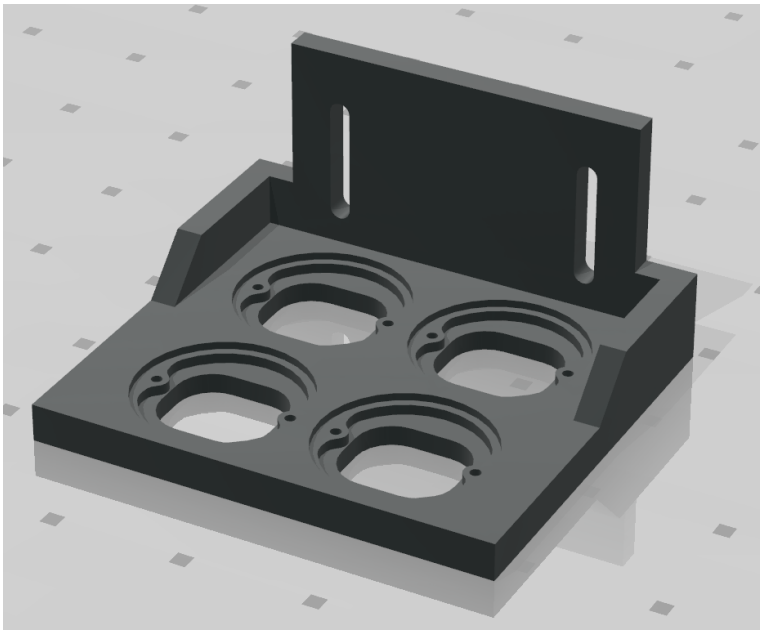

Figure 6.2: CAD model of the mounting bracket.

Each sensor is assigned a number and remains in the same position within the bracket for all measurements. Figure 6.3 shows a view of the test setup from above with the sensor numbers marked. Also shown are the sensor’s coordinate system in red, and the test rig’s coordinate system in green. To take advantage of the limited space, measurements are started in the top left corner of the liner (similar to what is shown in the figure).

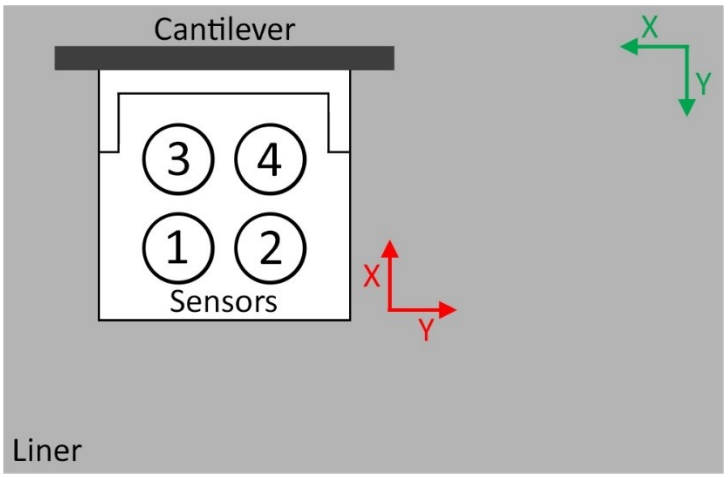

Figure 6.3: Experimental setup, viewed from above, showing sensor coordinate system (red) and test rig coordinate system (green).

The sensors are connected to a set of header pins soldered onto a PCB board by jumper wires. The PCB serves as a wiring junction, connecting the five SPI lines of each sensor so that only a single wire for each line (SCLK, MOSI, MISO, VIN, and GND) connected to the DUE. The SS lines are also routed through the junction. The DUE is connected via USB to a laptop. The entire setup is shown in Figure 6.4

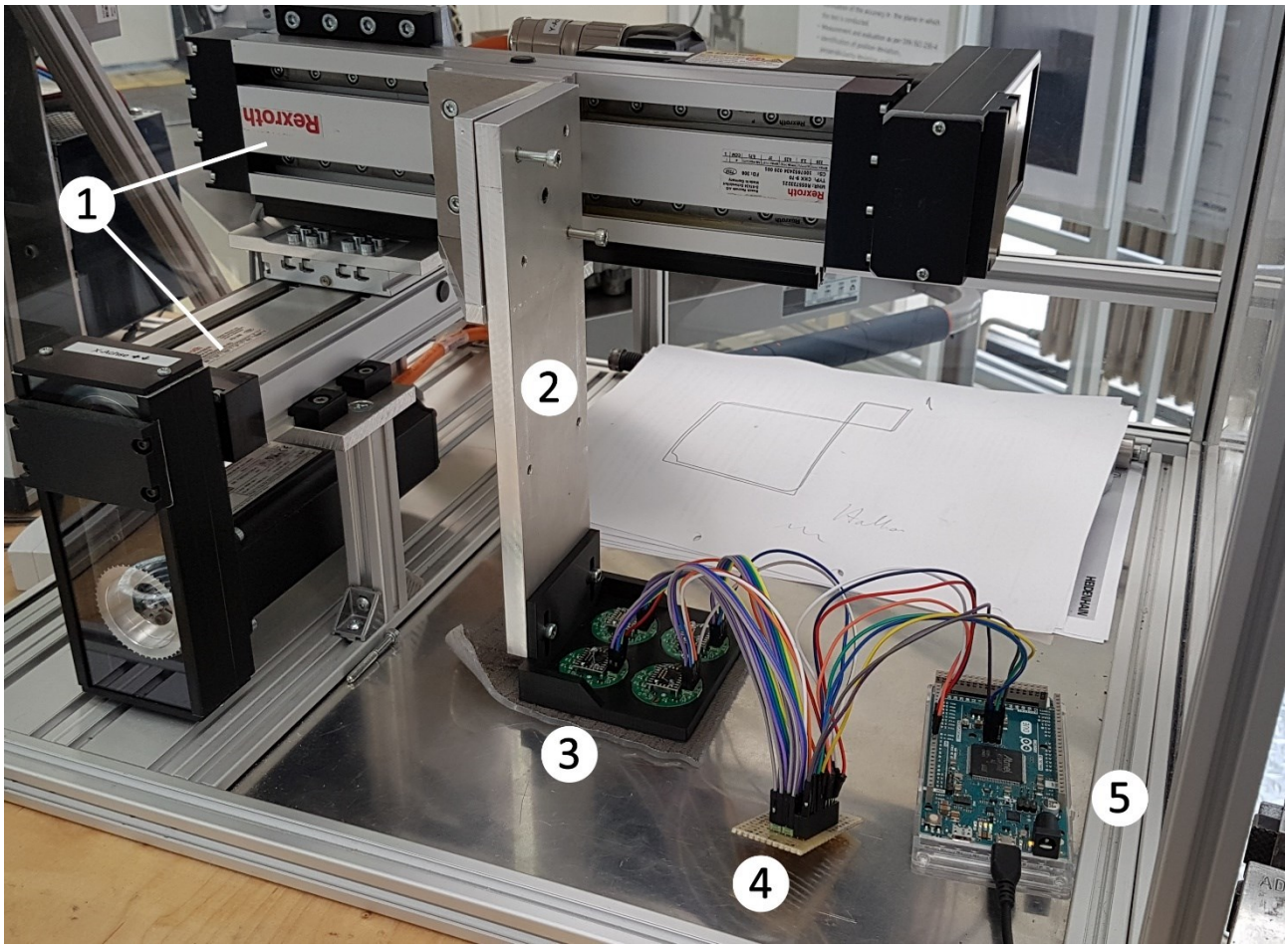

Figure 6.4: The test rig with linear drives (1), cantilever (2), test bracket, sensors and liner (3), wiring junction PCB (4), and Arduino Due (5) with USB cable leading to laptop.

The accuracy of the test rig is evaluated by performing ballbar tests with radii of 1 mm, 5 mm, and 10 mm at speeds of 1 mm/s, 10 mm/s, and 100 mm/s. Ten measurements per combination are taken with the bracket and sensors attached. Two sample paths are shown in Figure 6.5. Overall, the results are comparable to those presented by Somogyi; it is unclear whether he performed the tests with the bracket and sensor attached, but in either case, the addition of a larger bracket and three sensors appears to have had a minimal impact on the system's accuracy. Differences may also be due to the system having been disassembled and reassembled in the interim years. Values are presented and compared with his results in Table 6.1.

Table 6.1: Maximum deviations from ballbar tests in µm, mean and standard deviation for ten trials each. Values at right are from [27] for comparison.

		Speed in mm/s			Speed in mm/s	
		1	10	100	10	100
Radius in mm	1	3.3439 ± 0.1878	7.3437 ± 0.0687	7.4195 ± 0.1185	5.6452	7.2612
	5	1.7561 ± 0.1501	5.5877 ± 0.1672	14.2141 ± 0.2965		
	10	1.3687 ± 0.0872	5.1194 ± 0.1648	12.6919 ± 0.2087	5.1229	13.2264

Although Somogyi performed these tests, he did no theoretical analysis to determine the level of impact the test rig's inaccuracies might have on the measurements; such an analysis will be presented here. At the maximum resolution of 8 200 cpi, one count is equal to 3.1 µm. The liner used requires a calibration factor of approximately 2 [28], meaning that in practice the sensor's maximum sensitivity is around 6.2 µm. As the maximum deviations seen in the ballbar tests for 10 mm/s and 100 mm/s are of the same scale (5.1194 – 7.4195 µm) or greater (12.6919 – 14.2141 µm), the possibility of the test rig's inaccuracies affecting measurement results cannot be discounted. However, in practice it is not possible to quantify the errors for the measurements performed and such errors are neglected during the analysis.

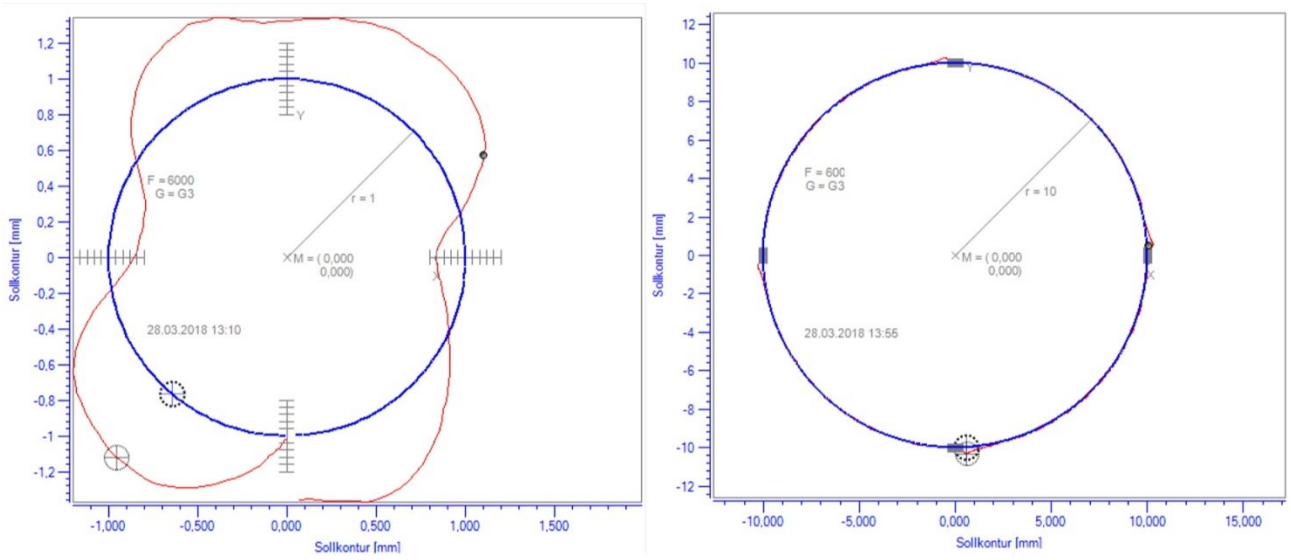

Figure 6.5: Sample ballbar test plot. Left: 1 mm radius at 100 mm/s, 7.47144 µm maximum deviation; Right: 10 mm radius at 1 mm/s, 5.1382 µm maximum deviation.

A further source of systematic error is the liner. It was used during Somogyi's original system assessment in 2015 as well as the subsequent ARP experiments in 2016 and is no longer in mint con-

dition (Figure 6.6). The liner's name and number, written in permanent marker, cover about 1/3 of the textile side and, although faded, are still visible. The material itself is somewhat distorted, as evidenced by the curving weave. The thickness varies by approximately 1 mm across a diagonal, resulting in a corresponding change in distance between liner and the sensors. Finally, over the course of the measurements performed for this work, a corner of the mounting bracket began to snag on fibers of the textile, pulling these up and causing significant fraying in one corner. Given the consistent location of sensors within the bracket, limited available space, and constraints of either repeating or not repeating given paths across the material, the individual sensors were exposed to these inhomogeneities to varying degrees. The extent of their impact was not explicitly evaluated; no obvious deviations within the measurements were noted as being likely due to the fraying or permanent marker, but the inconsistent weave orientation and changing separation distances are likely to have influenced results. This will be discussed in more detail in the appropriate analyses.

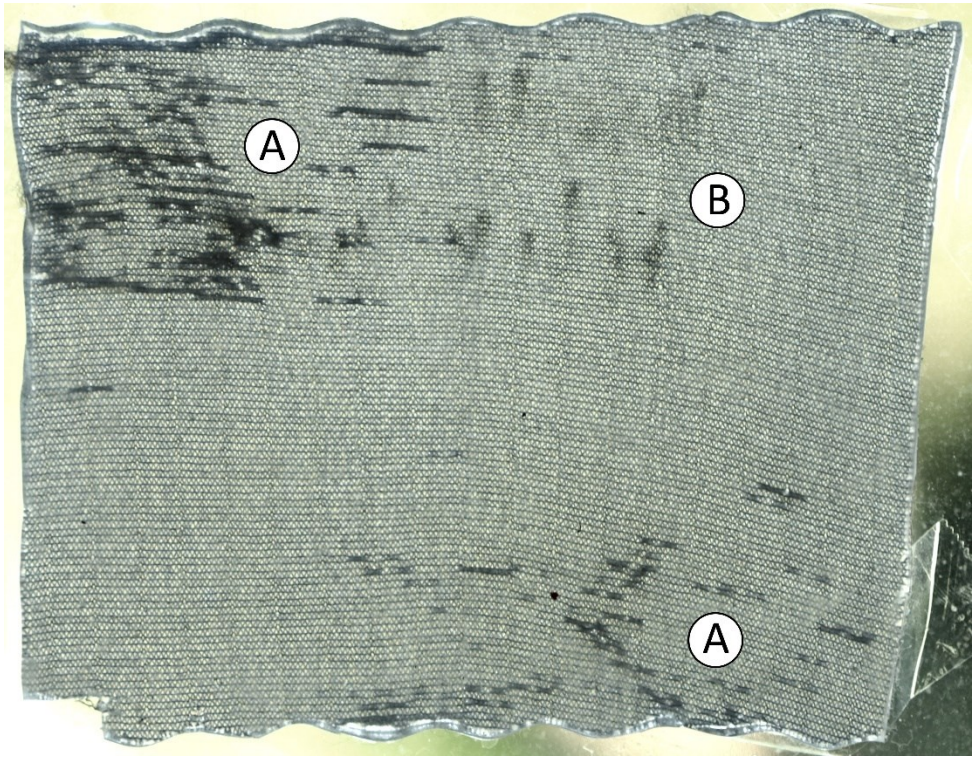

Figure 6.6: Backlit liner, contrast enhanced slightly to show: A) locations of fraying; B) location of permanent marker labelling (partially obscured by the fraying). Distortion in the weave is also visible. Photograph taken after completion of measurements with Canon EOS Rebel T6i with Canon 100mm macro lens at f/5.6, 1/100s, ISO 200.

Two measurement sessions ae conducted: one to evaluate the calibration process, and one to evaluate performance over sample paths. For each trial, firmware is uploaded and the sensor registers confirmed. SPI clock frequency, baud rate, and transmission rate per sensor were set to 14 MHz, 115 200 bps, and 200 Hz. Tested distances and speeds are chosen to correspond to those used in previously published work to allow results to be more easily compared. The chosen distances are 1 mm, 5 mm, 10 mm, and 40 mm; the chosen speeds are 1 mm/s, 10 mm/s, and 100 mm/s. Tests

were conducted in both the proper sensor-liner alignment (orientation 1) and with the liner rotated 90° (orientation 2).

6.2 Calibration

These measurements are taken using the system's built-in calibration function. The GUI signals the sketch that a calibration will be performed instead of a measurement, and the sketch enters calibration mode. The calibration is performed over a fixed distance specified in the GUI. Each direction is calibrated separately, with the PC sending start and stop signals before and after the distance is traversed. The start signal zeros the counts in the given direction. While the sensor is in motion, the counts accumulate. The stop signal ends count acquisition and the DUE sends the total number of counts to the PC, where it is converted into mm and then compared with the given distance to determine the calibration factor k_i. This calculation is shown by example of the X direction in Equation 5.1, where k_x is the calibration factor, c_x are the counts measured and Δx the specified distance in mm. No additional information (e.g. SQUAL-value or times) is obtained.

$$k_x = \frac{\frac{25.4\,\frac{\text{mm}}{\text{in}}}{8200\ \text{cpi}} * c_x}{\Delta x} \qquad \text{Equation 5.1}$$

Two series of 15 measurements for each combination of distance, speed, and orientation ae taken. In the first series (local), all measurements are taken over the same paths on liner. In the second series (regional), each measurement is taken over different paths on the liner. Factors obtained from the local series will indicate the repeatability of measurements over the same path and thus the maximum accuracy and precision of the system; factors obtained from the regional series will indicate performance over the liner surface as a whole. Comparison of the two will indicate the relative influence of the liner's microstructure.

Of 1440 dimensionless calibration factors measured, 34 were identified as outliers with the aid of boxplots and removed. The removed points are detailed in Table 6.2. A factor was considered an outlier if its value was above 2.5 or below 1.6. Two high outliers (6.5885 and 4.1389) were recorded for the first two regional readings of sensor 1 in X direction over 1 mm at 1 mm/s. The remaining 32 values were low (range 0.1412 – 1.3564). Of these, 22 occurred in X direction with 15 of those being from the regional series. The opposite was true in the Y direction, in which 8 of 10 outliers occurred during the local series. Orientation 1 had more outliers than orientation 2 (26 and 6, respectively).

For all measurement conditions, the majority of outliers (24) occurred during the first or second measurements and affected all four sensors. Somogyi noted a similar tendency for the first one or two calibration measurements to be erroneous. The fact that all four sensors are often affected at the start of measurements suggests an unknown systematic cause. This was not investigated further,

as the calibration process itself does not take much time and the results are immediately available for scrutiny; should such obvious errors occur, the calibration can simply be repeated.

Table 6.2: Removed calibration outliers. Distances in mm, speeds in mm/s; high values (H) > 2.5, low values (L) < 1.6.

Direction	Orientation	Series	Distance	Speed	Sensors	Measurement #	Value
X	1	Local	10	100	1,4	6, 8, 9	L
		Regional	1	1	1, 2	11	L
				10	1, 2, 3, 4	1	L
			10	100	1, 2, 3, 4	2	L
	2	Local	1	100	1, 2, 3	1	L
		Regional	1	1	1	1, 2	H
					2, 3, 4	1	L
Y	1	Local	10	1	1, 2, 3, 4	1, 2	L
		Regional	10	100	1, 4	2	L

Boxplots of the obtained factors, outliers removed, are shown for orientation 1 in Figure 6.7 and Figure 6.8. The corresponding boxplots for orientation 2 can be found in Appendix A2.

The values obtained are smaller than those presented by the ARP (Figure 3.10). A visual inspection shows that the local series provided more precise results (the boxes and whiskers are often invisible), and that differences between speeds are likely to be minor, while differences between sensors and distances are more readily apparent.

The statistical significance of these differences is investigated by applying the Wilcoxon Rank Sum test to settings groups. This test determines if there is a significant difference between the medians of the two tested groups, does not assume a normal distribution, and can be applied to groups of unequal sizes. As the test will be applied repeatedly, a Bonferroni correction is used to reduce the chance of a Type-1 error (incorrectly rejecting the null-hypothesis that the medians are the same).

The differences between speeds for a given sensor and distance, between distances for a given sensor number and speed, and between sensors for given distance and speed are investigated with a significance factor $\alpha = 0.0083$ (distances and sensors) or $\alpha = 0.0167$ (for speeds) to account for repeating the test six times within each grouping. Differences between local and regional series as well as orientations 1 and 2 are also considered ($\alpha = 0.05$). The resultant p-values may be found in the Digital Appendix.

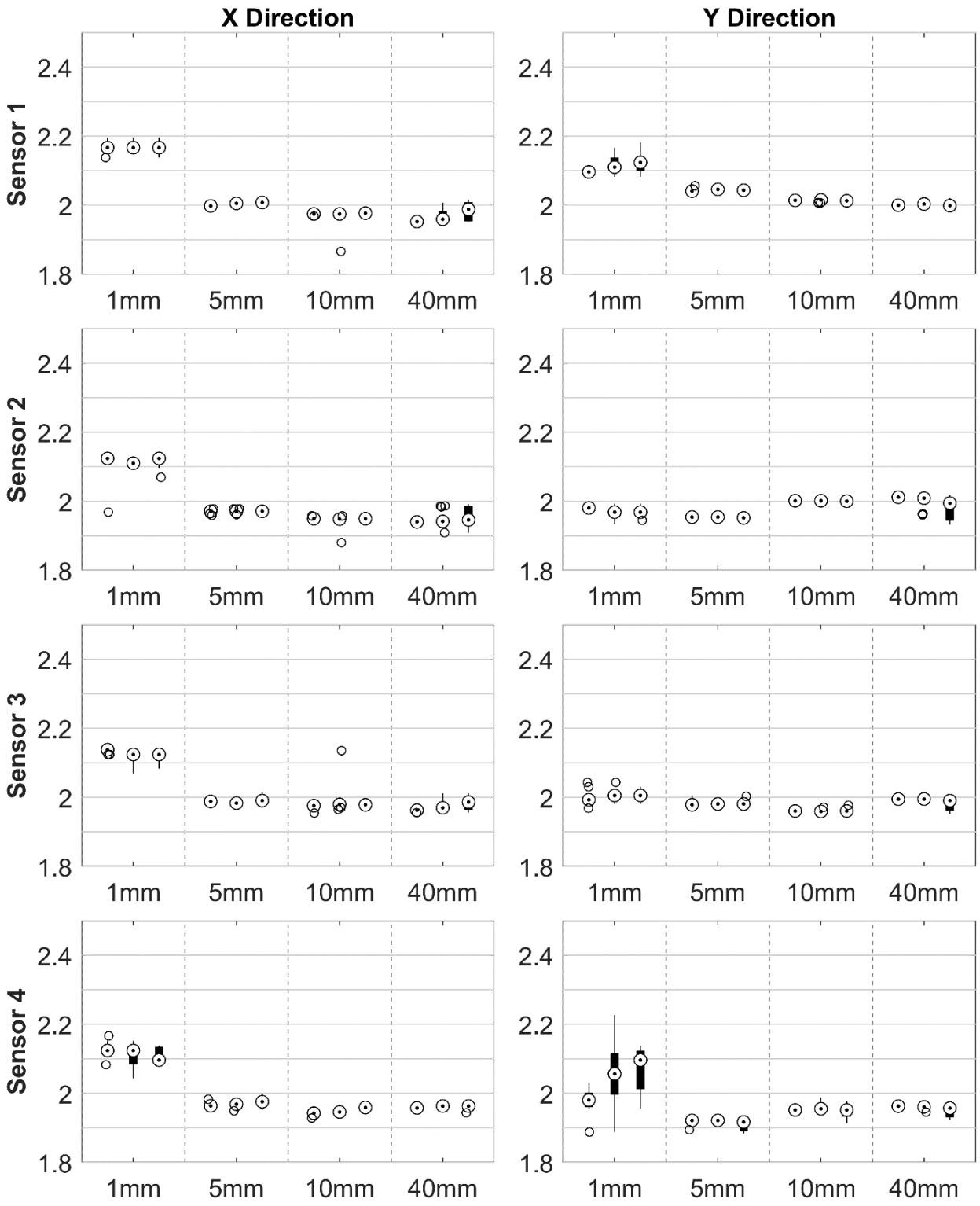

Figure 6.7: Orientation 1, local calibration factors. Speeds are shown in ascending order within each distance grouping. Each series consists of 15 measurements.

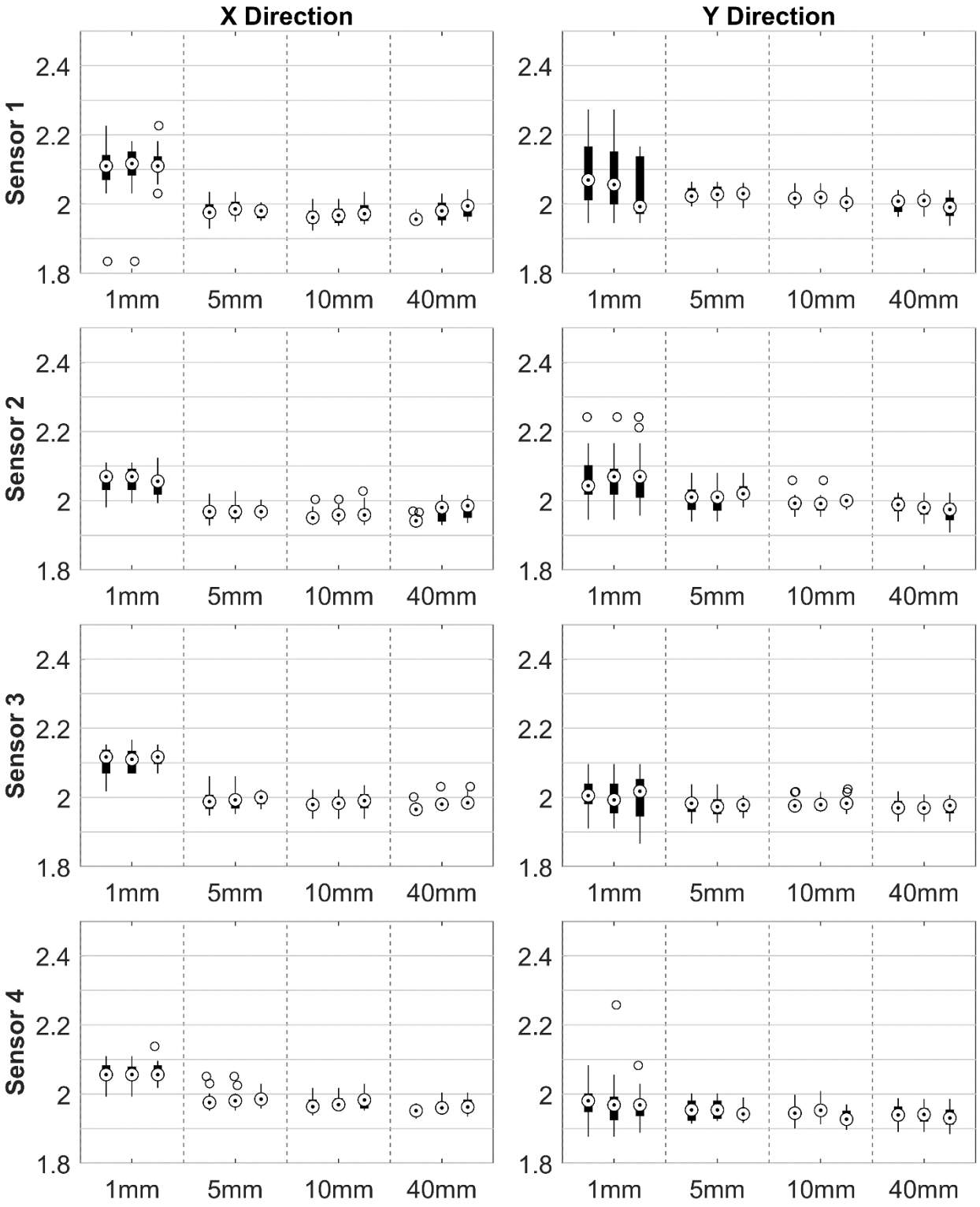

Figure 6.8: Orientation 1, regional calibration factors. Speeds are shown in ascending order within each distance grouping. Each series consists of 15 measurements.

The local series shows mixed statistical significance (some combinations of distance and sensor yielded a statistically significant result, others did not) between speeds for both orientations and directions. There are clear statistical differences between the distances and sensors. The regional

series showed no statistically significant differences between speeds or between distances, with the exception of 1 mm in X direction for both orientations, which was significantly different from the other distances. The differences between sensors showed mixed significance with the difference generally being between sensors 1 and 4. The comparison between series yielded mixed results. For X direction in both orientations, there was a statistically significant result at the 1 mm distance for all speeds and most sensors; otherwise, there was no discernable pattern to the significant results. Comparison by orientation yielded significant results between all groupings.

Based upon these findings, it can be said that, over a repeated path, the sensor is exceedingly precise and able to differentiate between different distances and speeds. This sensitivity largely disappears for measurements over non-repeating paths. The variation in values is correspondingly greater. The most variation and only significant difference in results was found to be between 1 mm and the greater distances. This indicates that the sensor is influenced by the local microstructure over sufficiently short distances. The fact that speeds are not shown to be a significant factor agrees with the qualitative statement made in [18] and the findings in [27] (here, a difference between speeds was observed but of no statistical significance).

The importance of the macro texture is also evident in the difference between orientations. Greater variation was seen in the Y direction for both orientations than in the X direction, again in accordance with the findings reported in [18] and indicating a lower sensitivity in this direction. Differences by sensor were inconsistent and it is unclear if they are due to the sensors themselves requiring different factors or due to their different offsets from the liner caused by its non-uniform thickness. The latter hypothesis is supported by the fact that, when a difference was noted between the sensors, it is almost always between sensors 1 and 4. Being on opposing corners of the bracket and given the orientation of the bracket relative to the liner, sensor 1 always experienced the smallest offset while sensor 4 experiences the greatest; the offsets for sensors 2 and 3 are qualitatively similar.

Calibration factors averaged by speed can be found for local and regional measurements in both directions in Appendix A3.

6.3 Accuracy and Precision

Measurements over sample paths are taken to evaluate the system's accuracy and precision and to determine which calibration settings yield the best-suited factors. Tests are run over three paths: square, linear, and diagonal. For each path, one measurement consisting of 15 repetitions is taken for every combination of distance, speed, and orientation. Each repetition of the square path covers the same portion of liner. Due to the size constraints imposed by the liner, linear paths measured in 5 mm and 10 mm increments partially repeated themselves, while those measured in 40 mm increments are always repeated. Diagonal paths do not repeat themselves for any sampling distance. Full sensor data (X and Y displacements, SQUAL-value, and times) are recorded; no calibration factors are applied at time of measurement. A MATLAB script is used to identify the areas of motion and

extract the initial and final uncalibrated locations for each motion increment. The unscaled distances are used to determine the ideal calibration factor as shown in Equation 5.1 in the preceding section. Boxplots of these factors are shown in Figure 6.9-Figure 6.11 for orientation 1. The plots for orientation 2 are in Appendix A4. In some cases, measurement data is incomplete due to its last sections not being properly saved on the PC, typically for the longer measurement times as are required by e.g. 40 mm at 1 mm/s (duration > 10 minutes). In cases where the number of datapoints differ from the expected, the actual number is noted in parentheses in the figure description. Tabulated values of speed- and distance-averaged ideal calibration factors are given in Appendix A5.

A visual inspection of the plots reveals similar behavior to that demonstrated in the previous section. The comparably high precision of the square paths is likely due to each pass covering the same section of liner. Precision on the 40 mm linear path is likely high for the same reason. As the 5 mm and 10 mm distances each repeated only portions of their paths, the exact influence of these repetitions is unclear. Variation was greatest for the diagonal paths, which were not repeated at any distance.

As before, variation in the Y direction appears greater than in X direction. Differences between sensors is also apparent to varying degrees. Sensor 1 tends to have the highest factors, sensor 4 often the lowest; sensors 2 and 3 tend to require the approximately the same factors. This again suggests that inter-sensor differences may be more due to differences in offset than systematic causes: a smaller calibration factor corresponds to a greater number of counts over the same area, and it is conceivable that a greater offset would allow the sensor to count more features. This reasoning was also used by Somogyi to explain why his measurements using greater offsets had greater relative errors.

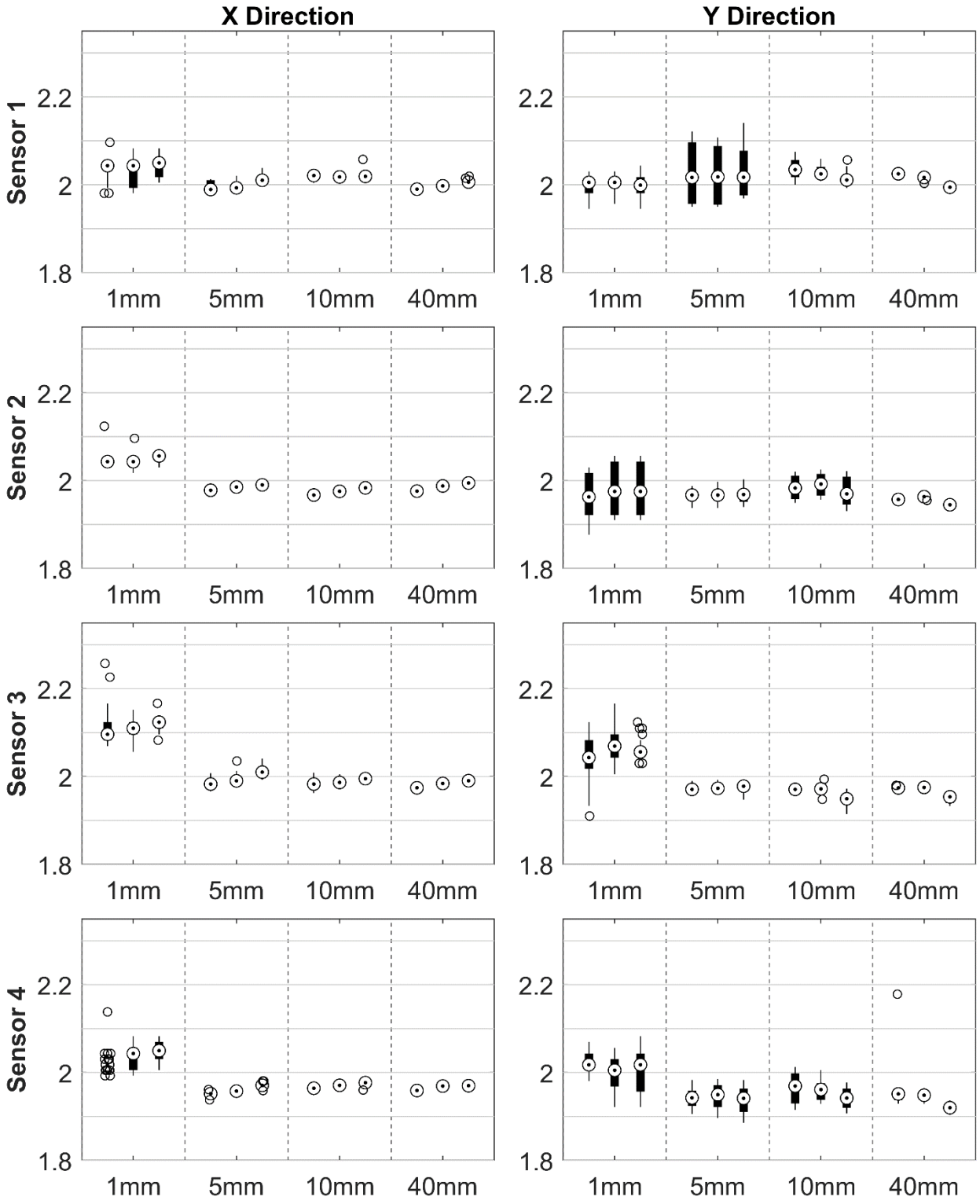

Figure 6.9: Orientation 1, square paths - ideal calibration factors. Speeds are shown in ascending order within each distance grouping. 30 samples each except 10 mm at 1 mm/s (24), 40 mm at 1 mm/2 (10), and 40 mm at 10 mm/s (28).

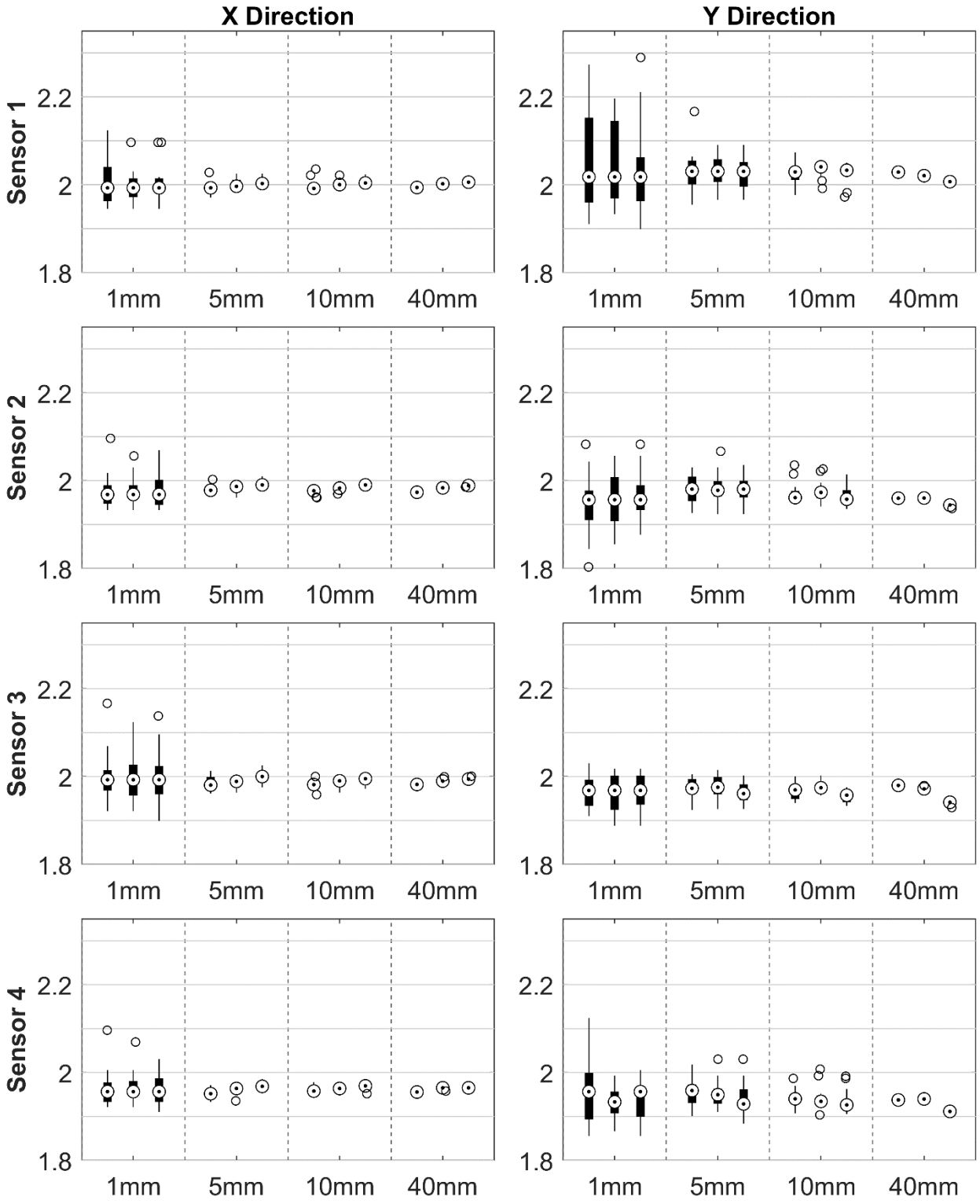

Figure 6.10: Orientation 1, linear paths - ideal calibration factors. Speeds are shown in ascending order within each distance grouping. 15 samples each except Y direction at 40 mm, 1 mm/s (6) and 40 mm, 100 mm/s (14). 15 samples each except X direction 40 mm at 1 mm/s (7) and Y direction 10 mm at 1 mm/s (14) and 40 mm at 1 mm/s and 10 mm/s (13).

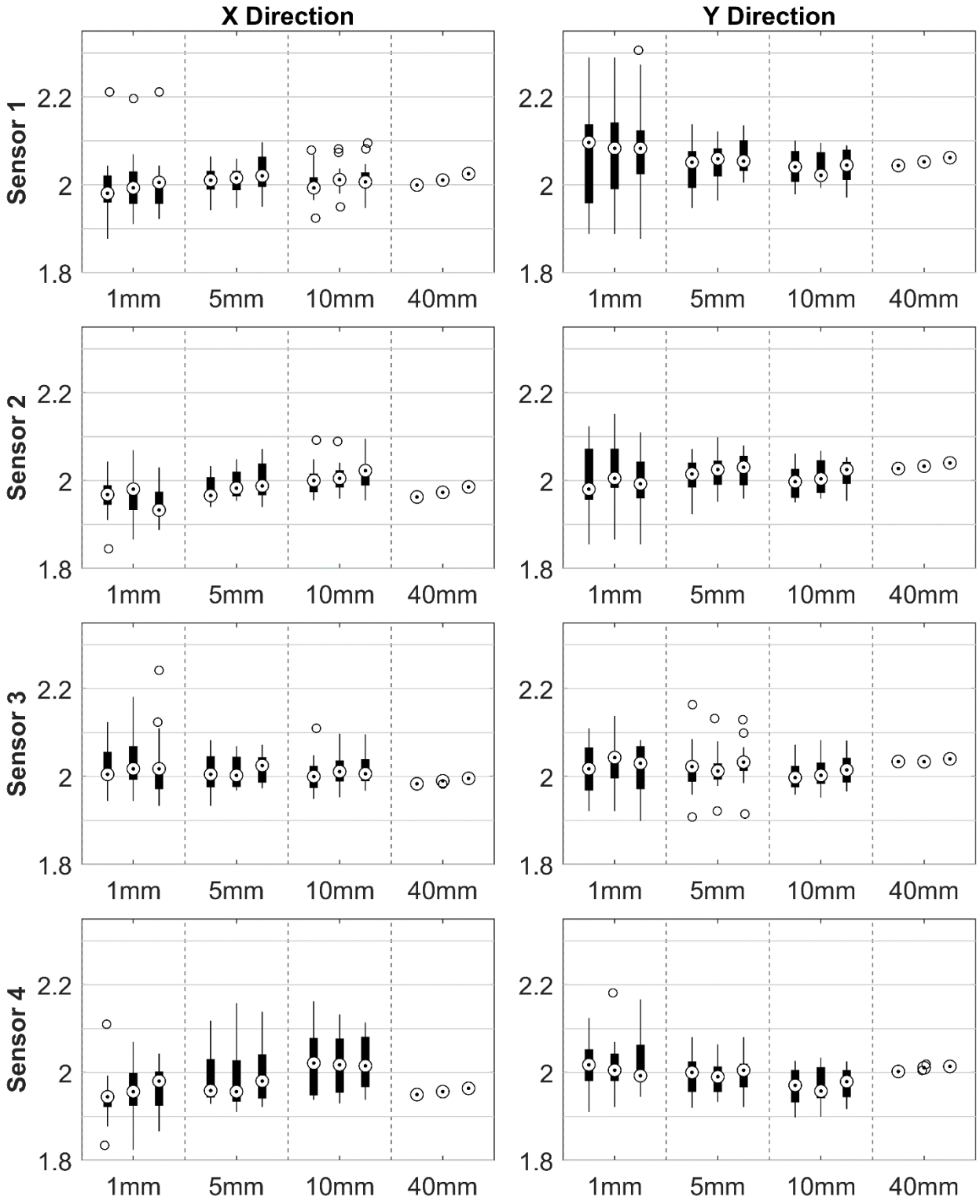

Figure 6.11: Orientation 1, diagonal paths - ideal calibration factors. Speeds are shown in ascending order within each distance grouping. 15 samples each.

Since the generally expected sensor behavior was established in the preceding section, the interest here is to determine if the ideally required factors differ based on the trajectory and which calibration factors offer the “best” results. As speed was previously identified as not being a significant

factor in regional situations, ideal factors for each sensor and distance are averaged by speed to simplify the analysis. The Wilcoxon Rank Sum test is applied to determine any difference between path types based on distances and sensors ($\alpha = 0.0167$); the resultant p-values are tabulated in the Digital Appendix. Results are inconclusive. For orientation 1, the X direction showed a statistically significant difference between square paths and the rest, while the Y direction showed differences between the diagonal paths and the rest. For orientation 2, differences between all methods were found for the X direction while the Y direction presented mixed results between square paths for 1 mm and 5 mm distances and more significant differences between all paths for 10 mm and 40 mm.

It is possible that these results are further confirmation of the influence of texture upon the measurements. The square paths likely differentiate themselves due to measuring the same sections of liner; higher variations of the diagonal paths may correspond to findings in [50] where similar behavior was noted.

To examine the measurements' accuracy and precision, an appropriate calibration factor must be selected from those determined in the preceding section. In practice, all sensors will be calibrated simultaneously using the same settings. To reflect this, each sensor's mean factor from a single setting choice will be applied to all paths. Since the factor's dependence upon surface texture is reduced for regional calibrations and for longer distances, factors resulting from the regional calibration over 40 mm were chosen. This choice is considered justified as a comparison of the plots of relative errors between all factors showed this setting to yield the overall most accurate results; a more rigorous selection process was not undergone because to do so when using the measurement system in practice would be impractical and the varied nature of the problem argues against tuning the selection too precisely.

The chosen factors are applied to the extracted motion increments through multiplication and the relative error of each calculated according to Equation 6.1. The mean and standard deviations of the relative error are determined for each distance, sensor, and path type combination (speeds are combined). The bias of the mean is interpreted as the accuracy, the magnitude of the standard deviation as the precision.

$$err_{rel} = \frac{(x_1 - x_0) - \Delta x}{\Delta x} \qquad \text{Equation 6.1}$$

The results are plotted in Figure 6.12-Figure 6.14 and will be discussed in detail below for orientation 1. Plots for orientation 2 are in Appendix A6.

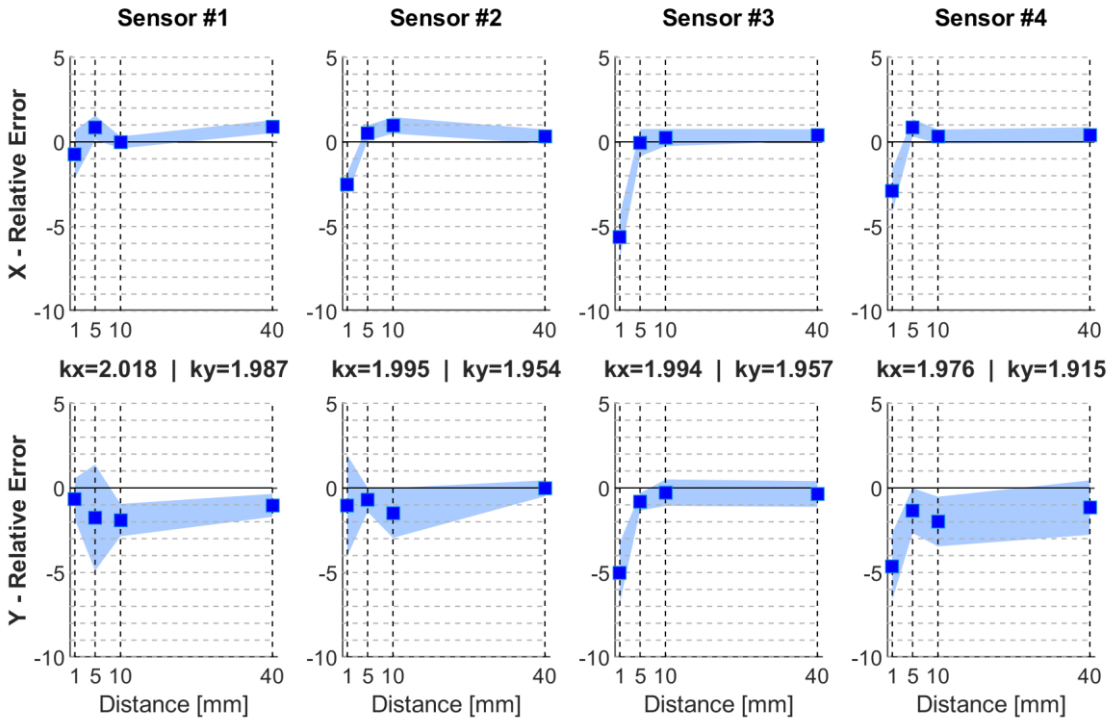

Figure 6.12: Orientation 1, square paths – relative errors (speeds combined) in %. Mean (markers) and standard deviation (shaded areas), with application of speed-averaged 40 mm regional calibration factors.

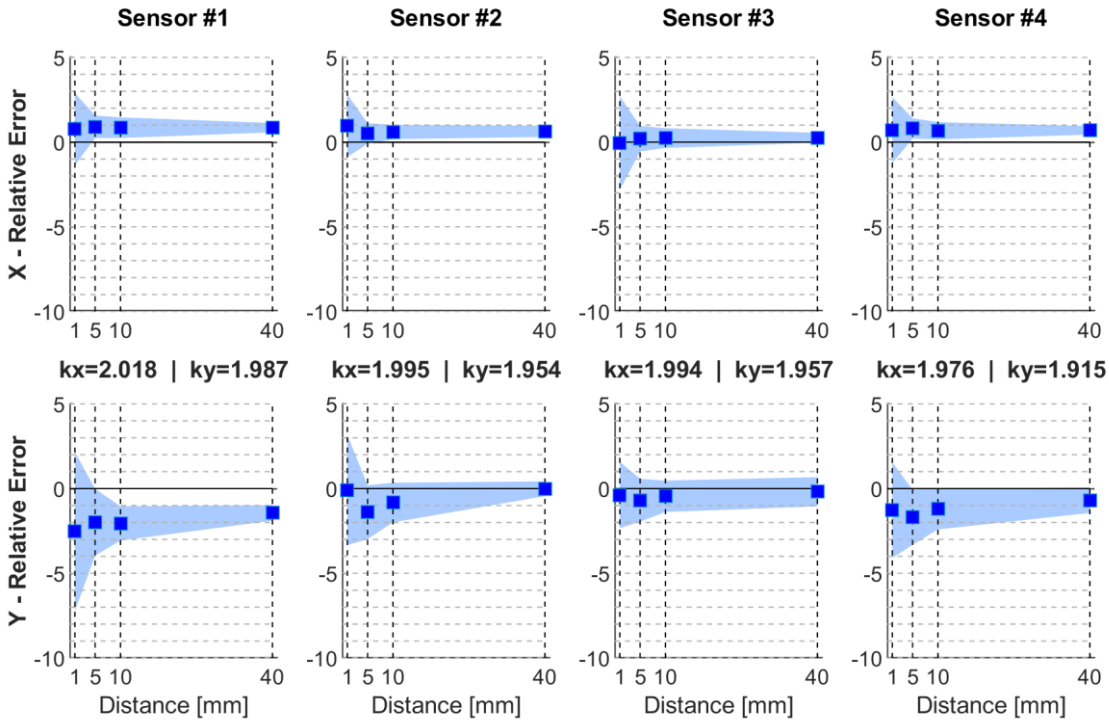

Figure 6.13: Orientation 1, linear paths – relative errors (speeds combined) in %. Mean (markers) and standard deviation (shaded areas), with application of speed-averaged 40 mm regional calibration factors.

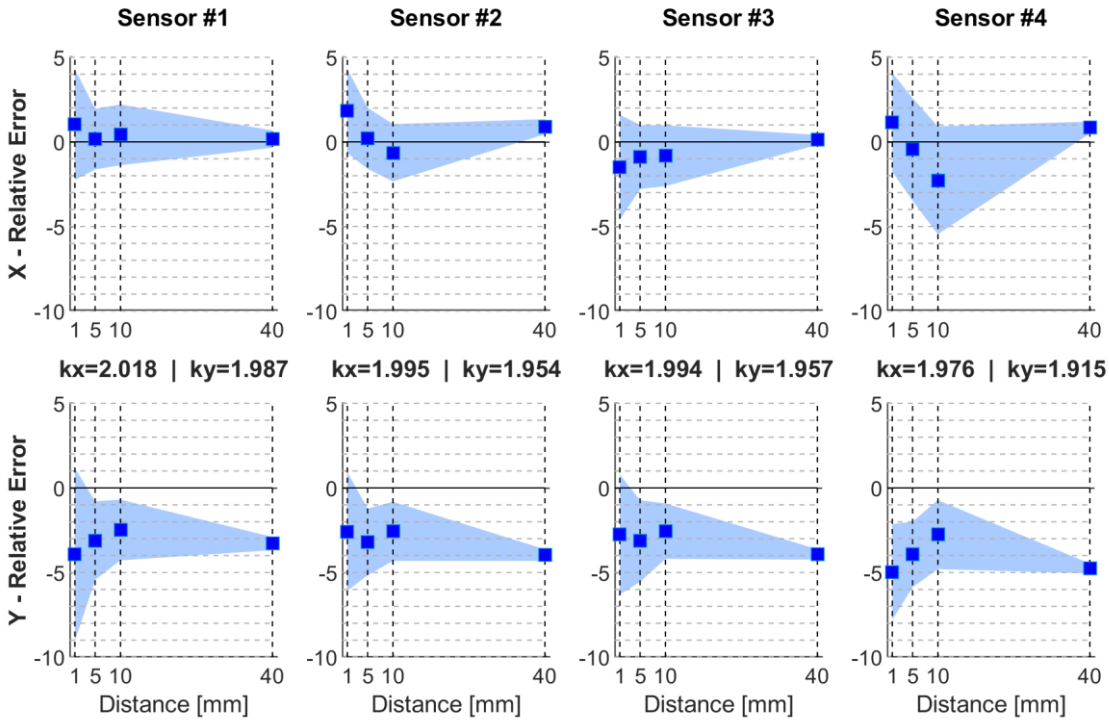

Figure 6.14: Orientation 1, diagonal paths – relative error (speeds combined) in %. Mean (markers) and standard deviation (shaded areas), with application of speed-averaged 40 mm regional calibration factors.

The results are encouraging. Overall, the chosen calibration factors achieved a bias within 2% of the true value except for smaller distances on the square paths and the Y direction for the diagonal paths. In the former case, the means drop sharply for 1 mm. This is likely due to the influence of microstructure revealed in Section 6.2: the particular 1 mm square being measured may have a texture that differs significantly from the average seen in larger squares. These local effects are exaggerated due to not having sampled any other areas and are not as pronounced for the linear and diagonal paths. In the latter case, the chosen factors systematically underestimate the Y displacement for the diagonal paths. Once again, this is likely a texture related effect similar to that documented in [50] where it was suggested that the images acquired during diagonal motion are not necessarily linear combinations of the respective X and Y motions.

Precision is also excellent. As expected, it is best for the square paths due to repeatedly measuring the same surface (< 1%). The linear paths showed good performance with curves resembling those presented by Noll in [18]: precision decreasing for smaller distances and more variation in the Y direction. A similar pattern is evident for the diagonal paths, although the precision here is less and accuracy is more dependent upon distance (10 mm either being a peak, or values alternating in relative magnitude). Orientation 2 behaves in a comparable manner. The maximum means and standard deviations between all sensors at 1 mm and 40 mm are shown in Table 6.3 for orientation 1. The corresponding table for orientation 2 is in Appendix A7. It is interesting to note that the extreme values for both orientations are most likely to be from either sensor 1 or sensor 4.

Table 6.3: Orientation 1 - maximum means and standard deviations of the relative error at 1 mm and 40 mm in %; number of sensor where maximum occurred in parentheses.

Path	Direction	Maximum at 1 mm		Maximum at 40 mm	
		Mean	Standard Deviation	Mean	Standard Deviation
Square	X	-5.6137 (1)	1.4073 (1)	0.8999 (1)	0.4572 (4)
	Y	-5.0141 (3)	2.9712 (2)	-1.1638 (4)	1.6097 (4)
Linear	X	0.9617 (2)	2.7693 (3)	0.8511 (1)	0.2579 (2)
	Y	-2.5300 (1)	4.6938 (1)	-1.4345 (1)	0.8628 (3)
Diagonal	X	1.8268 (2)	3.2794 (1)	0.8949 (2)	0.5115 (1)
	Y	-4.9682 (4)	5.1152 (1)	-4.7649 (4)	0.3896 (1)

6.4 Sensor Drift

A drift in the data is apparent when the X and Y displacements are plotted. Sample trajectories from one measurement are shown in Figure 6.15 (detail) and Figure 6.16 (entire measurement). At first glance, the behavior appears similar to that encountered during the ARP, but, in this case, it cannot be attributed to the sensor physically slipping over the liner surface. A closer inspection of the curves shows disturbances in the stationary direction while the other is in motion (e.g. disturbances to X displacement while sensor is moving in Y direction). It is possible that the drift may be due to a misalignment between sensors and the test rig's cantilever. While this cannot be excluded as a contributing factor, if it were the only reason then one would expect that the drift approximately cancel itself out for the square paths – and this is not the case. The magnitude of the disturbance in one direction is greater than in the other. From a visual inspection of plots of all the measured paths, it is unclear if the drift is systematic or more random in origin.

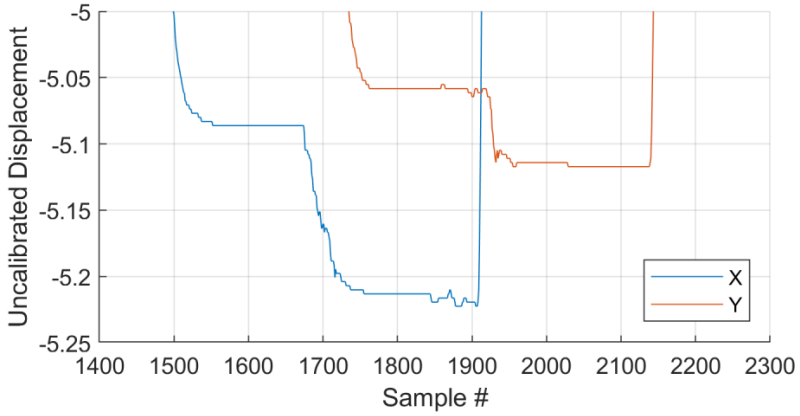

Figure 6.15: Detail of uncalibrated 10 mm square drift at 100 mm/s for sensor 3. Deceleration curve of test rig visible.

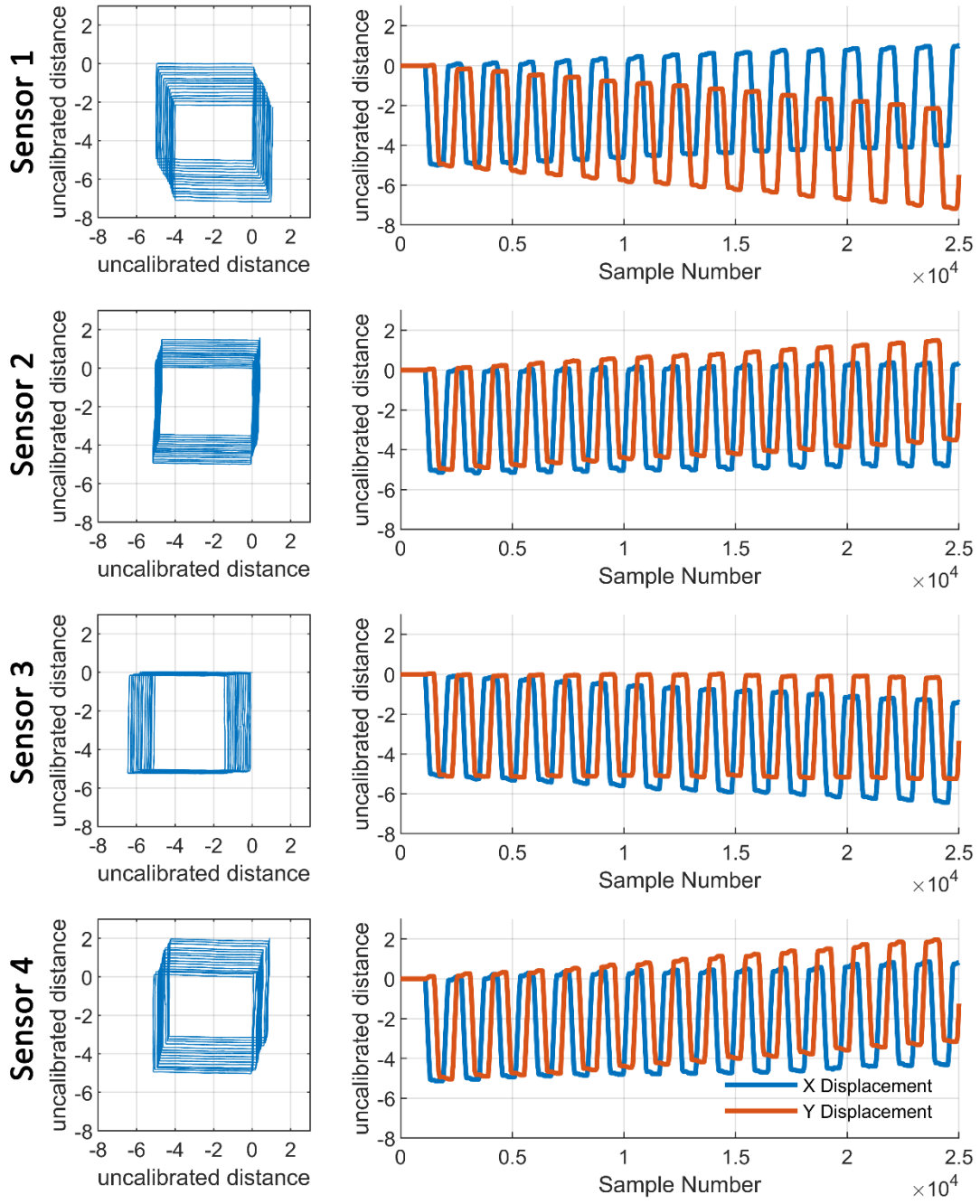

Figure 6.16: Uncalibrated 10 mm square paths at 10 mm/s. Orientation 1. The path taken as measured by the sensor (left), displacements plotted against sample number (right).

Since the disturbances occur almost exclusively within the time of motion along the other axis, it is suspected that the motion is in some way responsible. Only the square and linear paths are examined as the simultaneous motion of the diagonal path precludes a simple analysis. Data is left uncalibrated as doing so would introduce artificial scale differences. To determine the nature of the behavior, the disturbed sections and the corresponding increment of motion are isolated using the cuts made in the preceding section. The ratio of disturbance and motion displacements is then taken

for both X and Y (Equation 6.2) to determine the relative drift of one axis per unit traveled by the other:

$$\Delta x_{rel} = \frac{\Delta x_{disturbance}}{\Delta y_{motion}}$$

$$\Delta y_{rel} = \frac{\Delta y_{disturbance}}{\Delta x_{motion}}$$

Equation 6.2

The means and standard deviations of this ratio are taken over all speeds at each distance and plotted in Figure 6.17 and Figure 6.18 for orientation 1; the plots for orientation 2 are in Appendix A8. A positive value indicates that the drift occurs in the direction of motion while a negative value indicates that the drift is opposed to the direction of motion.

Overall, the results strongly indicate an approximately constant systematic drift of varying magnitudes for each sensor. Drift in the X direction occurs in the same direction as the accompanying Y motion, while drift in the Y direction is opposed to the accompanying X motion. Standard deviations increase for smaller distances, and deviations of the mean relative to those of the greater distances become more pronounced, especially for X drift on square paths. As was observed in previous sections, the microstructure has a strong influence over small distances and is likely to be the source of these changes. It must be noted that, while the drift appears to be relatively constant for the three greater distances, especially between 10 mm and 40 mm, this is only an assumption. The mean drifts over linear paths for orientation 1 are presented in Table 6.4.

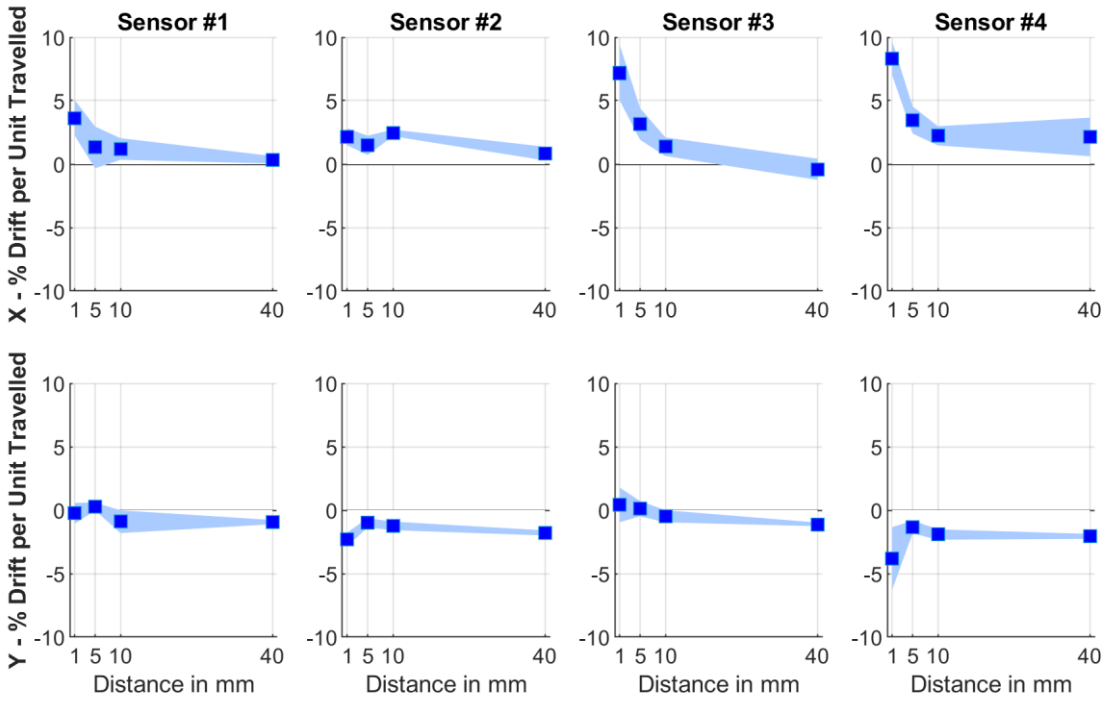

Figure 6.17: Orientation 1, square paths - mean and standard deviation of percent drift of stationary axis (X in first row, Y in second) relative to the moving axis.

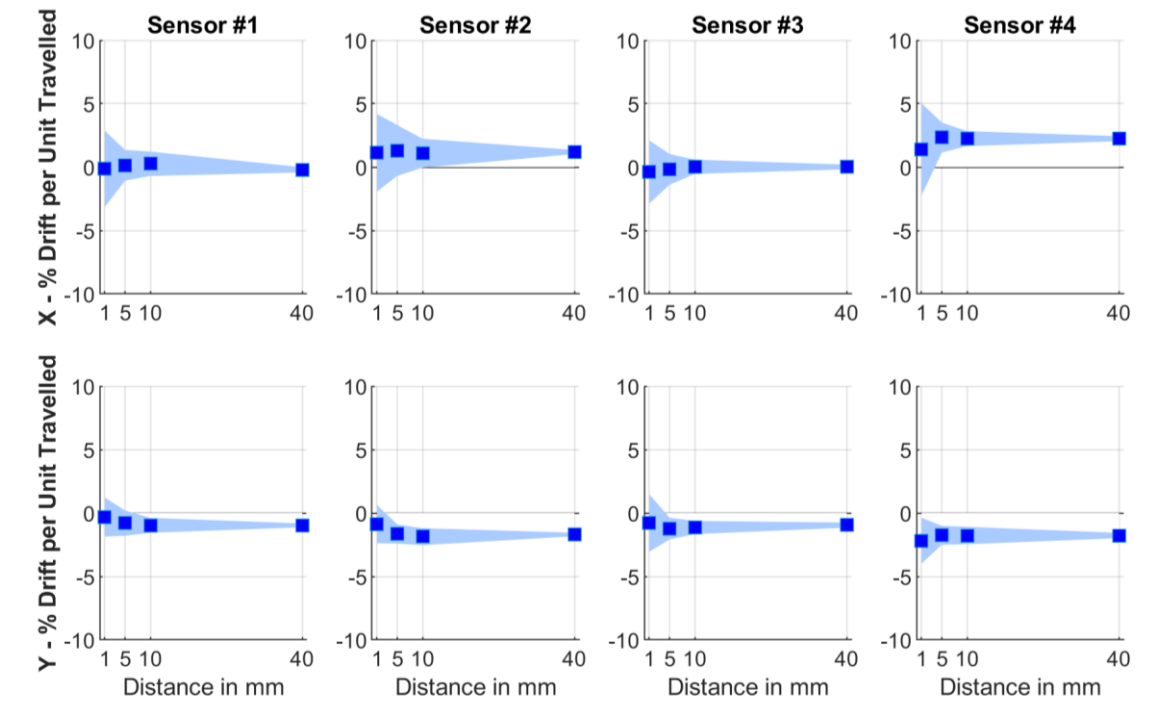

Figure 6.18: Orientation 1, linear paths - mean and standard deviation of percent drift of stationary axis (X in first row, Y in second) per unit travelled by the moving axis.

Table 6.4: Mean percent drift of stationary axis per unit travelled by the moving axis for linear paths. Orientation 1.

Sensor	Drift Direction	Distance in mm 1	5	10	40	Average
1	X	-0.1374	0.1370	0.2613	-0.2177	0.0108
	Y	-0.3204	-0.7928	-0.9713	-0.9718	0.7641
2	X	1.1295	1.3066	1.0933	1.1871	1.1791
	Y	-0.8760	-1.6584	-1.8642	-1.6892	1.5219
3	X	-0.3905	-0.1812	0.0246	0.0077	0.1348
	Y	-0.7800	-1.2272	-1.1439	-0.9512	1.0256
4	X	2.3963	2.3299	2.2418	2.2407	2.3022
	Y	-2.1768	-1.7563	-1.7739	-1.7740	1.8702

Averaged across all distances, sensor 1 displays the least amount of drift (0.0108% in X, 0.7641% in Y) while sensor 4 displays the most (2.3022% in X, 1.8702% in Y). As with the difference in calibration factor magnitude noted earlier, this suggests that the distance between sensor and surface is important for reliable performance. One plausible explanation for the drift is that, due to the surface texture and its distortion (as noted in 6.1), counts are erroneously accumulated for the stationary direction. Just as sensor 4 required a smaller calibration factor due to being higher off the sur-

face and thus able to count more features, this wider field of view also permits more false counts to be made.

Theoretically, it is possible to characterize the drift and remove it from the data; however, preemptively applying a correction to real data would be unwise – and, as the correction would likely be itself an averaged value for drift under laboratory conditions, it would not succeed in completely sanitizing the data. A better option would be to ensure a small, constant distance between sensor and liner and to counteract any recorded drift during data analysis.

As no analysis of the drift was performed for the diagonal paths, its extent during simultaneous motion along both axes is unknown.

7 The Measurement System

The final measurement system with sensor array is designed in preparation for a pilot study. Modifications have been made to the hardware and both the appearance and functionality of the MATLAB GUI for a more robust and user-friendly system. The system is configured to accept up to eight sensors, but this capacity can easily be expanded.

The ARDUINO DUE is housed in a custom-designed 3D-printed box. The box is generously sized to allow for the possible future wireless conversion of the system, which will be touched upon in Chapter 8. At 11.3×7.3×5.4 cm, the box is too large to fit in most pockets but can be carried in a fanny-pack or lightly modified to attach to a belt. The front panel allows access to the external power supply and both USB ports while also providing sockets for the sensor cables and a trigger signal. For easy identification, the SPI wires are given consistent colors throughout the system (both inside the box and in the sensor cables) and a different color is given to each of the eight SS wires inside the box as shown in Figure 7.1.

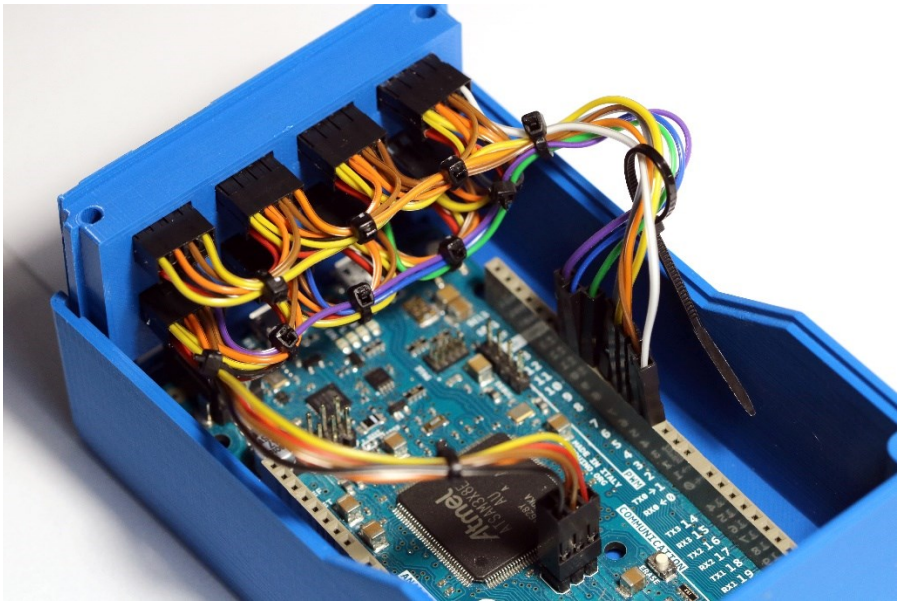

Figure 7.1: Detail of the wiring between the microcontroller box's front panel and the Arduino Due.

DuPont connectors have been used throughout the system to allow for greater flexibility and alleviate concerns about poorly soldered contacts. The sensor housing has been altered accordingly. Two versions were created based off the original design from the ARP. In the first version, the cable exits

the housing parallel to the socket wall; in the second, the cable exits the housing perpendicular to the socket wall[12] (Figure 7.2). The first version is recommended due to its lower profile.

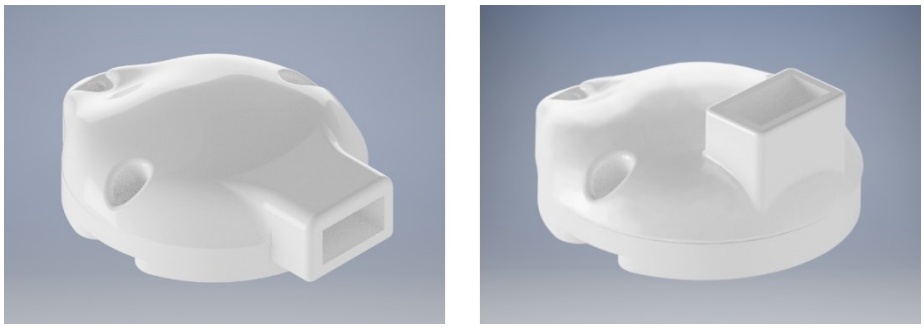

Figure 7.2: Modified sensor housings with cable exiting parallel (left) and perpendicular (right) to the socket wall.

As of this writing, four cables have been prepared with lengths of approximately 86.4 cm or 34 inches. The junctions with the connectors are protected by heat-shrink tubing (Figure 7.3). As the connectors are symmetrical 2×4 blocks, there is the possibility that they will be inserted the wrong way into either sensor or Due box. To prevent this, one of the two unused pin connections has been blocked, making it impossible to insert the cables incorrectly; visual markings will also be added.

Figure 7.3: Sensor cables with DuPont connectors at both ends.

Sensor and SPI settings are hard-coded into the sketch. The SPI clock frequency is set to 14 MHz; the resolution and frame rate are respectively set to 8 200 cpi and 12 000 fps. The baud rate is set to 115 200 bps and is hard-coded into both the sketch and the GUI.

The new MATLAB GUI is shown in Figure 7.4 and has been overhauled to provide a more comprehensive measurement experience. Available serial COM ports are automatically detected. The user

[12] This version was created and 3D-printed because of difficulties unsoldering the straight header pins from the sensor units. To use the recommended version, these would need to be removed and replaced with right-angle headers.

selects the number of sensors to be used and the transmission frequency per sensor. Uploading the firmware[13], confirming the sensor registers, and using a 5 V trigger signal are all optional. Calibration factors are stored in a dedicated file and entries are easily added, modified or removed. Calibration measurements may also be performed from a dedicated panel. A preliminary analysis may be performed to confirm a successful measurement. This consists of plotting the received data and calculating the transmission periods and frequencies. Data, including results of the analysis and metadata, may be saved to a location of the user's choice. The data is saved as a structure containing the various quantities as matrices or cell matrices, whichever is appropriate. More details on GUI functionality and the implemented data structure is given in user guide in the Digital Appendix. Simple error handling is incorporated throughout the GUI.

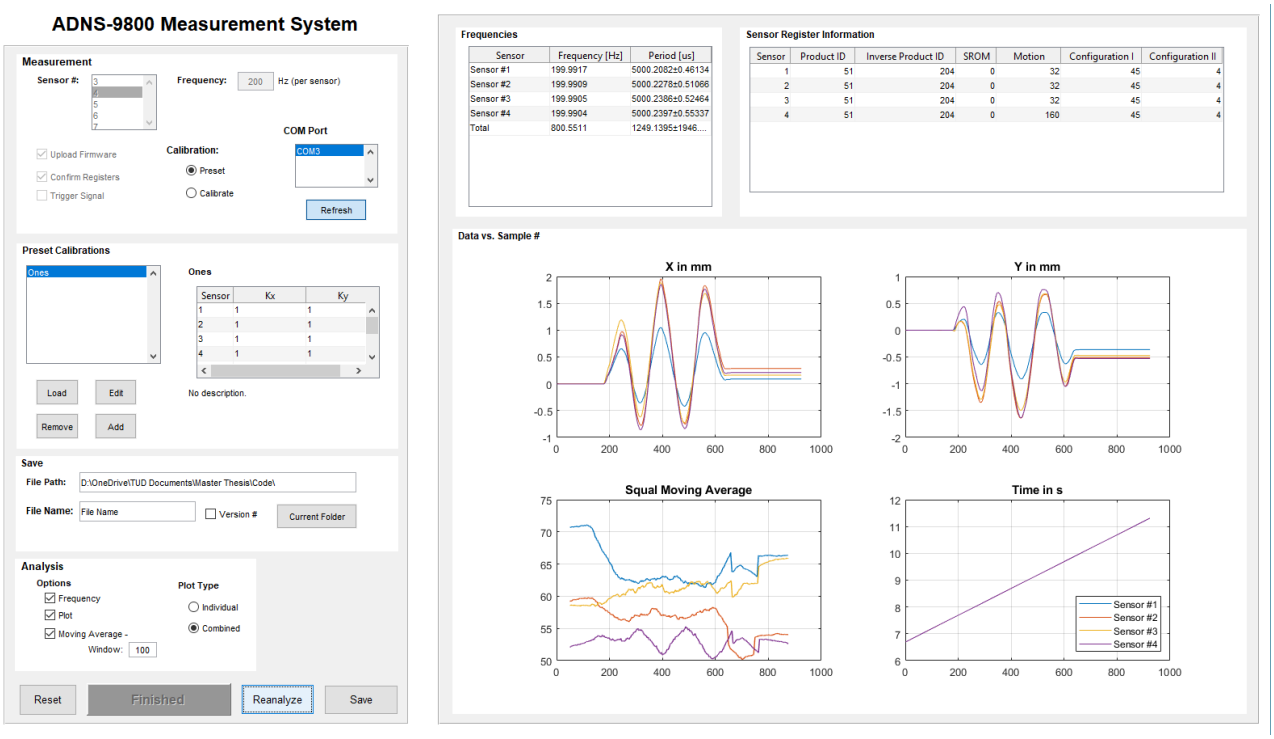

Figure 7.4: Screenshot of the revised GUI after the completion of a measurement using four sensors. Results are shown to the right, while the various settings are located to the left.

For appropriately chosen calibration factors, the system's accuracy and precision are excellent. Determining the best factors to use is, by nature of the variation of the liner surface, not a precise science. Based upon the results presented in Chapter 6, general guidelines can be given:

[13] Choosing not to upload firmware saves time on startup, but the effect of not doing so has not been formally investigated. Although no adverse effects were noted in informal experiments without uploading firmware, this option should be treated with appropriate caution.

- No statistically significant differences were found between the three speeds tested and thus any convenient speed may be used. As the influence of varying speed during calibration was not investigated, use of a constant speed is recommended.
- Calibration factors obtained over small distances are highly influenced by the local microstructure of the textile. For generally applicable results, calibrations should be performed over greater distances.
- X and Y directions should be calibrated separately due to the sensor's differing sensitivities.
- Calibrations should be carried out with the offset between sensor and liner being as closed the one expected in practice as possible.
- The analysis did not definitively determine whether sensors need to be individually calibrated. Although it appears that most inter-sensor differences are due to differing offsets, in which case the same factors could be used, this was not conclusively shown. Individual calibration is therefore recommended.

During validation of the system on the test rig, errors occurred only rarely and were of three types. For calibrations, there is a small chance of the initial few readings taken being unusually low. However, if one is familiar with the range of values to expect for a given surface, these deviations will be obvious and the faulty measurements can simply be discarded. For path measurements, a handful of random errors occurred. Out of 839 measurements[14], only seven errors were detected of this sort were detected. An example is shown in Figure 7.5. Finally, for measurements with durations approaching or exceeding 10 minutes, large sections of the data were not saved if the measurement was stopped too soon after the motion of interest ceased. Waiting a minute or two longer eliminated the problem and its source was not further investigated. It should be noted that, as the proposed measurement durations were two minutes, it is unlikely that this error will be encountered in a practical setting.

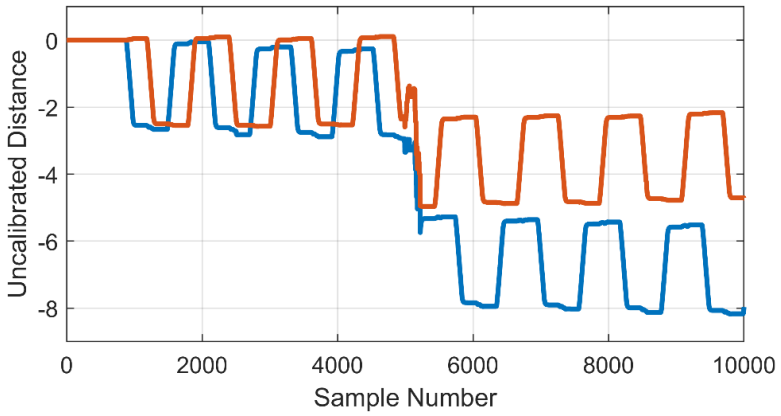

Figure 7.5: Example of a measurement error; sensor 4 over 5 mm distance at 10 mm/s.

[14] A measurement here being the displacement in a single direction for the given settings. So for a given orientation, path type, distance and speed, there are two measurements (X and Y displacements) for each of the four sensors. Some measurements in this total represent trials that were repeated due to a serious error.

8 Conclusions

The desired measurement system with sensor array has been successfully developed. The original system accommodated only one sensor and was reported to achieve maximum frequencies of only 30-64 Hz depending on the PC (and likely MATLAB version) used. Even at these low frequencies, random and inexplicable errors routinely occurred at the beginning of measurements. Replacement of the functional model's ARDUINO UNO with the more capable ARDUINO DUE in Chapter 4 and the subsequent revision of both ARDUINO sketch and MATLAB GUI in Chapter 5 resulted in dramatic improvements in performance. As MATLAB's serial read speeds were slower than the rate at which data was received, the terminal program REALTERM was introduced to circumvent the problem of the overflowing input buffer while still permitting measurements to be controlled from the GUI. Two versions of the sketch were created that send different quantities of data. For the experimentally evaluated 13-byte version, the system achieves 900 Hz on the PC hardware used. Running the 9-byte sketch, the system is capable of 1299 Hz. Errors occurred only rarely and had a localized impact within the measurement.

The experimental evaluation of the system presented in Chapter 6 built upon previously conducted experiments and gave further insight into the sensor's behavior on the chosen liner material. The sensors were shown to be sensitive to both the microstructure and macrostructure of the liner's textile surface. The dominance of the microstructure's influence over the tested 1 mm distances was evidenced by the significantly different results obtained for these measurements. The influence of the macrostructure (texture) was seen in the variation in results depending on sensor-liner orientation and the observed drift. The effect of the offset between sensor and liner was also apparent, particularly in the differing behaviors of sensors 1 and 4 between which was the greatest difference in offset. Sensor 4, with the greatest offset, required a smaller calibration factor and showed drifts of greater magnitude; the opposite was true of sensor 1 with the smallest offset. This behavior suggests that a greater offset allows a sensor to view a larger area and thus to count more features. Despite these differences, none of the sensors showed an obvious tendency to be more or less accurate than the others.

With appropriately chosen calibration factors, the system's accuracy and precision were both better than previously reported and easily met the requirement of a maximum absolute error of 1 mm over a 40 mm distance. The X direction displayed better performance than the Y direction. For the correct sensor-liner alignment, the greatest biases of the mean and largest standard deviations were 0.90% and 0.51% in the X direction and –4.76% and 1.61% in the Y direction[15]. These values stem from different sensors and paths but are presented together to indicate the worst recorded performance. Discounting the large biases for Y measurements along diagonal paths caused by a systematic underestimation of the displacement due to the calibration factor chosen, the system satisfies the

[15] The large bias is due to the chosen calibration factors systematically underestimating the Y displacement for diagonal paths. Better performance was seen for linear paths where the greatest bias was –1.43%.

requirement of a maximum relative error of 1 mm over a 40 mm distance (this corresponds to a 2.5% relative error).

The resultant measurement system was then presented in Chapter 7. Hardware and software were overhauled for optimum performance and to accommodate a total of eight sensors. Recommendations for calibrating the system were also given.

There are also a few aspects of system behavior that could be further investigated. The role of PC hardware has not been formally addressed thus far nor has the system been tested on other PCs. Although the impact of PC hardware on the measurements themselves has been minimized as much as possible by using REALTERM, instead of the more computationally intensive MATLAB, to read and save the incoming data, there are likely to still be differences in performance.

The observed sensor drift was only evaluated in the simple case of motion in one direction. Given the observed differences in results between linear and diagonal motion, it is likely that this behavior differs somewhat when the sensor is moving diagonally. As it is unlikely that the sensor will undergo an entirely unidirectional motion in practice, it is important to better understand the sensor's behavior in this situation.

More work could also be done to definitively determine the importance of the SQUAL-value. Although no formal evaluation was performed for this work, qualitative observations supported Somogyi's findings that the SQUAL-value depends upon the sensor's location on the liner.

The calibration factors used when evaluating the relative error were the average results of 45 measurements. A recommended minimum number of calibration measurements to be performed for a satisfactory result was not given, and currently only single measurements may be taken through the GUI. At minimum, an expansion of the calibration function should be undertaken to allow for automatic averaging of multiple readings.

In its current form, the system is ready to be used in a pilot study. However, it is still a wired system and the necessary tether between subject and PC imposes limits on its use. Although the ARDUINO DUE is not itself capable of wireless communication, additional components may be incorporated to achieve this functionality. One possible solution is XBEE modules produced by DIGI INTERNATIONAL, INC., which are used for this purpose by another student at the institute. XBEEs are radio modules that connect to the serial ports of the communicating devices. They are rated for ranges up to 60 m indoors (standard version) and over-the-air baud rates of up to 250 000 bps [71], which are more than sufficient for this application. The microcontroller box is large enough to accommodate both a wireless module and the battery necessary to power the system.

Appendix

A1. Additional Material for Section 5.4.2

Table A1.1: The times required for various string reading and saving configurations in MATLAB. Ten repetitions of 1000 values for reading one digit or ten digits, mean and standard deviation shown.

		Time in µs	
Operation		**1 digit**	**10 digits**
send string, read string, save as matrix/cell	`a=fscanf(s,'%s'); xa=str2num(a); x=[x xa];`	257.1 ± 2.7	1028.5 ± 2.6
	`a=fscanf(s,'%s'); x=[x str2num(a)];`	257.1 ± 4.8	1028.7 ± 2.6
	`a=fscanf(s,'%s'); x(end+1)=str2num(a);`	256.98 ± 7.7	1028.6 ± 1.6
	`a=fscanf(s,'%s'); x{end+1}=a;`	257.0 ± 2.2	1028.7 ± 2.8
	`x{end+1}=fscanf(s,'%s');`	257.0 ± 3.0	1028.1 ± 2.0
send string, read float, save as matrix	`a=fscanf(s,'%f'); x=[x a];`	257.0 ± 5.8	1028.3 ± 2.5
	`x=[x fscanf(s,'%f');`	257.1 ± 1.7	1028.3 ± 2.0
	`x(end+1)=fscanf(s,'%f');`	257.0 ± 1.9	1028.7 ± 3.4

A2. Boxplots of Calibration Factors for Orientation 2

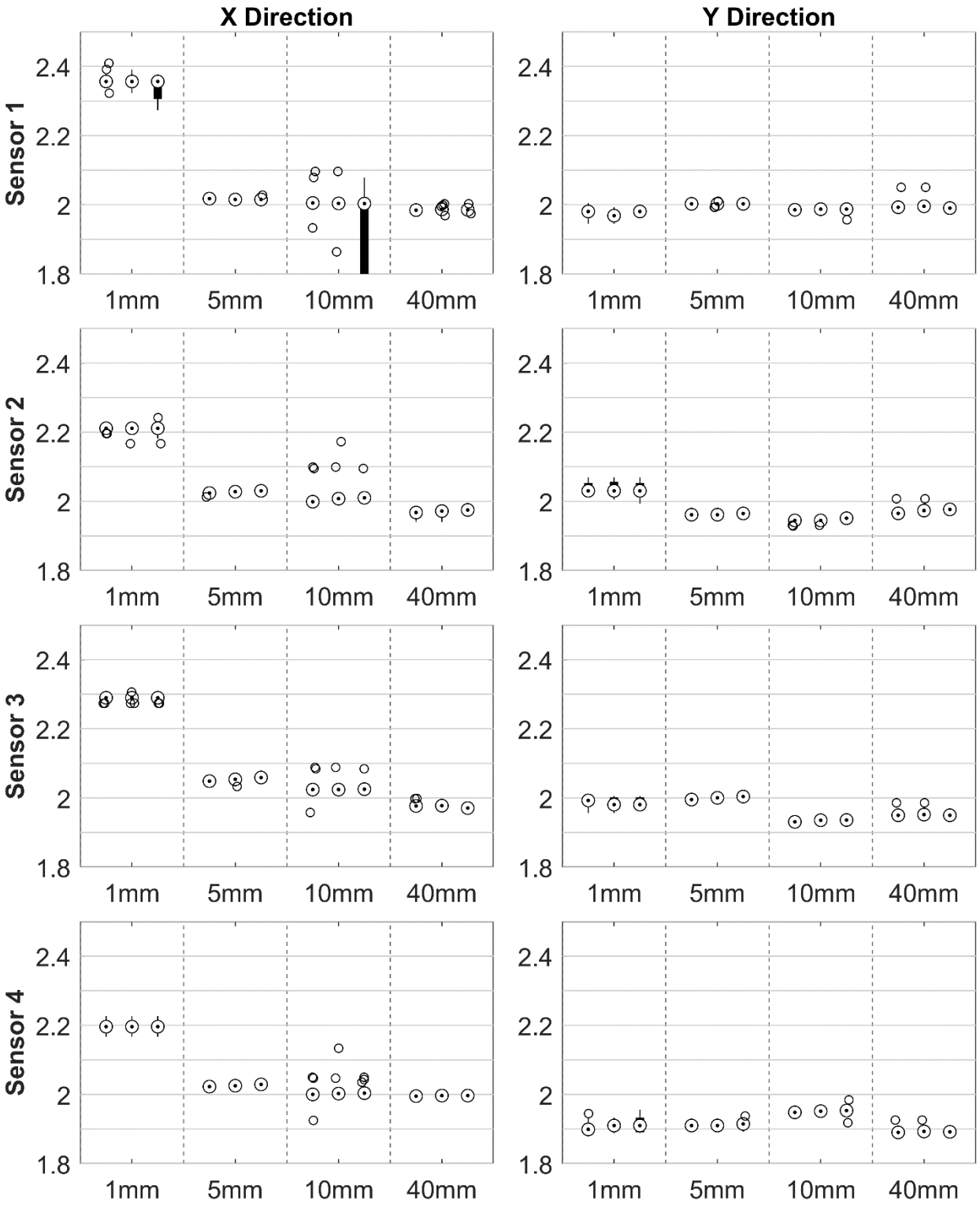

Figure A2.1: Orientation 2, local calibration factors. Speeds are shown in ascending order within each distance grouping. Each series consists of 15 measurements.

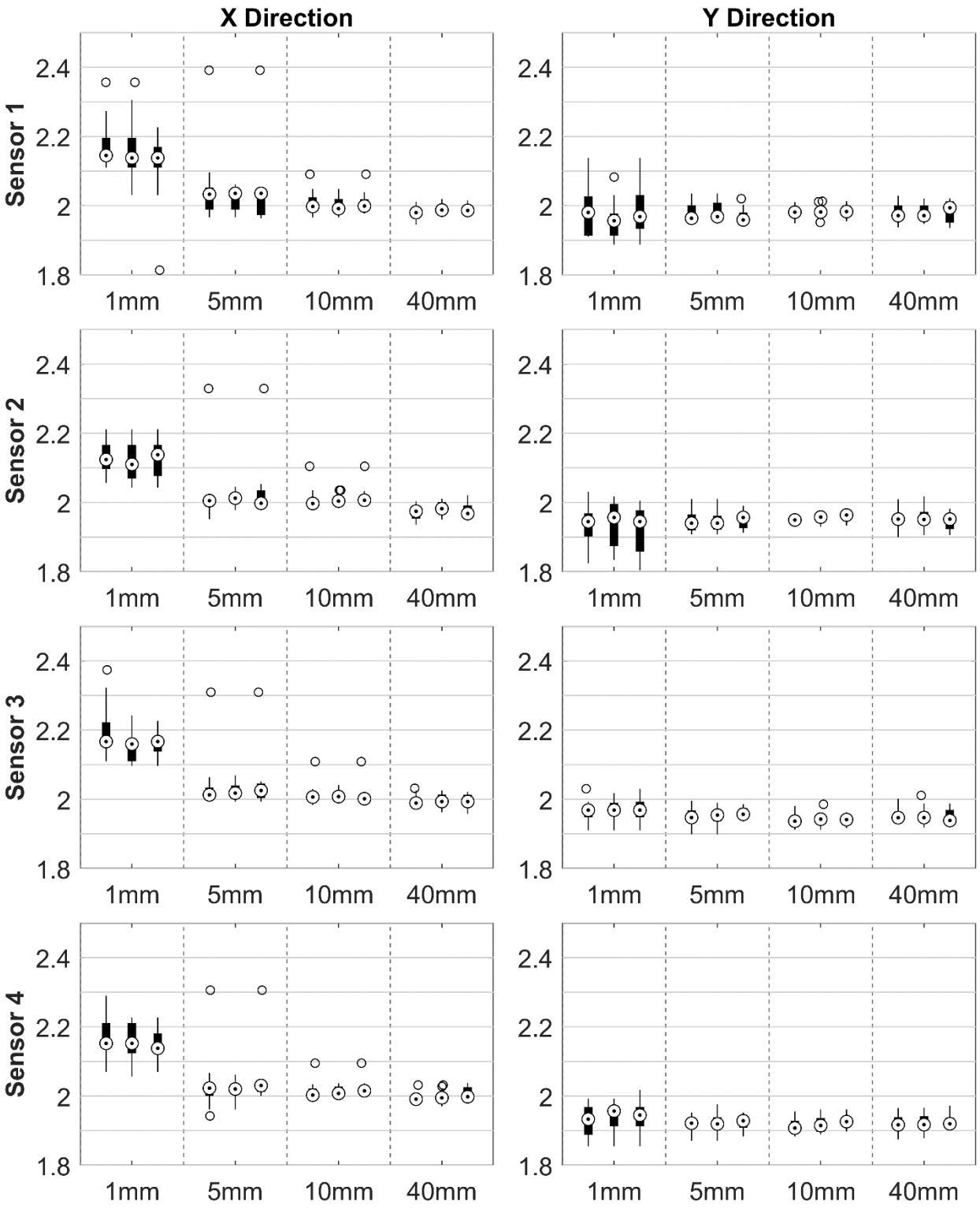

Figure A2.2: Orientation 2, regional calibration factors. Speeds are shown in ascending order within each distance grouping. Each series consists of 15 measurements.

A3. Speed-Averaged Calibration Factors for both Orientations

Table A3.1: Orientation 1 calibration factors averaged over speed, distances given in mm.

Method & Direction		Distance	Sensor 1	Sensor 2	Sensor 3	Sensor 4
Local	X	1	2.1409	2.1170	2.1380	2.0910
		5	2.0137	1.9676	1.9952	1.9857
		10	1.9721	1.9556	1.9843	1.9596
		40	2.0119	1.9870	1.9957	1.9654
	Y	1	2.1315	1.9742	2.0144	2.0992
		5	2.0513	1.9423	1.9749	1.8783
		10	2.0131	1.9940	1.9642	1.9284
		40	1.9900	1.9418	1.9598	1.9274
Regional	X	1	2.1122	2.0623	2.1224	2.0785
		5	1.9937	1.9776	1.9988	1.9915
		10	1.9810	1.9685	1.9964	1.9790
		40	2.0182	1.9950	1.9939	1.9761
	Y	1	2.0743	2.0561	1.9972	2.0016
		5	2.0311	1.9995	1.9794	1.9475
		10	2.0062	1.9756	1.9737	1.9325
		40	1.9874	1.9541	1.9573	1.9153

Table A3.2: Orientation 2 calibration factors averaged over speed, distances given in mm.

Method & Direction		Distance	Sensor 1	Sensor 2	Sensor 3	Sensor 4
Local	X	1	2.3219	2.2115	2.2929	2.2163
		5	2.0104	2.0362	2.0626	2.0318
		10	1.8895	2.0118	2.0310	1.9498
		40	1.9824	1.9850	1.9666	2.0043
	Y	1	1.9814	2.0187	1.9847	1.9480
		5	1.9937	1.9748	2.0102	1.9236
		10	1.9860	1.9552	1.9372	1.9585
		40	1.9901	1.9812	1.9439	1.8959
Regional	X	1	2.1263	2.1315	2.1658	2.1389
		5	2.0193	2.0199	2.0294	2.0273
		10	2.0013	2.0103	2.0088	2.0190
		40	1.9792	1.9928	1.9926	2.0022
	Y	1	2.0029	1.9410	1.9685	1.9317
		5	1.9829	1.9589	1.9596	1.9278
		10	1.9879	1.9533	1.9433	1.9239
		40	1.9825	1.9555	1.9517	1.9225

A4. Boxplots of Ideal Calibration Factors for Orientation 2

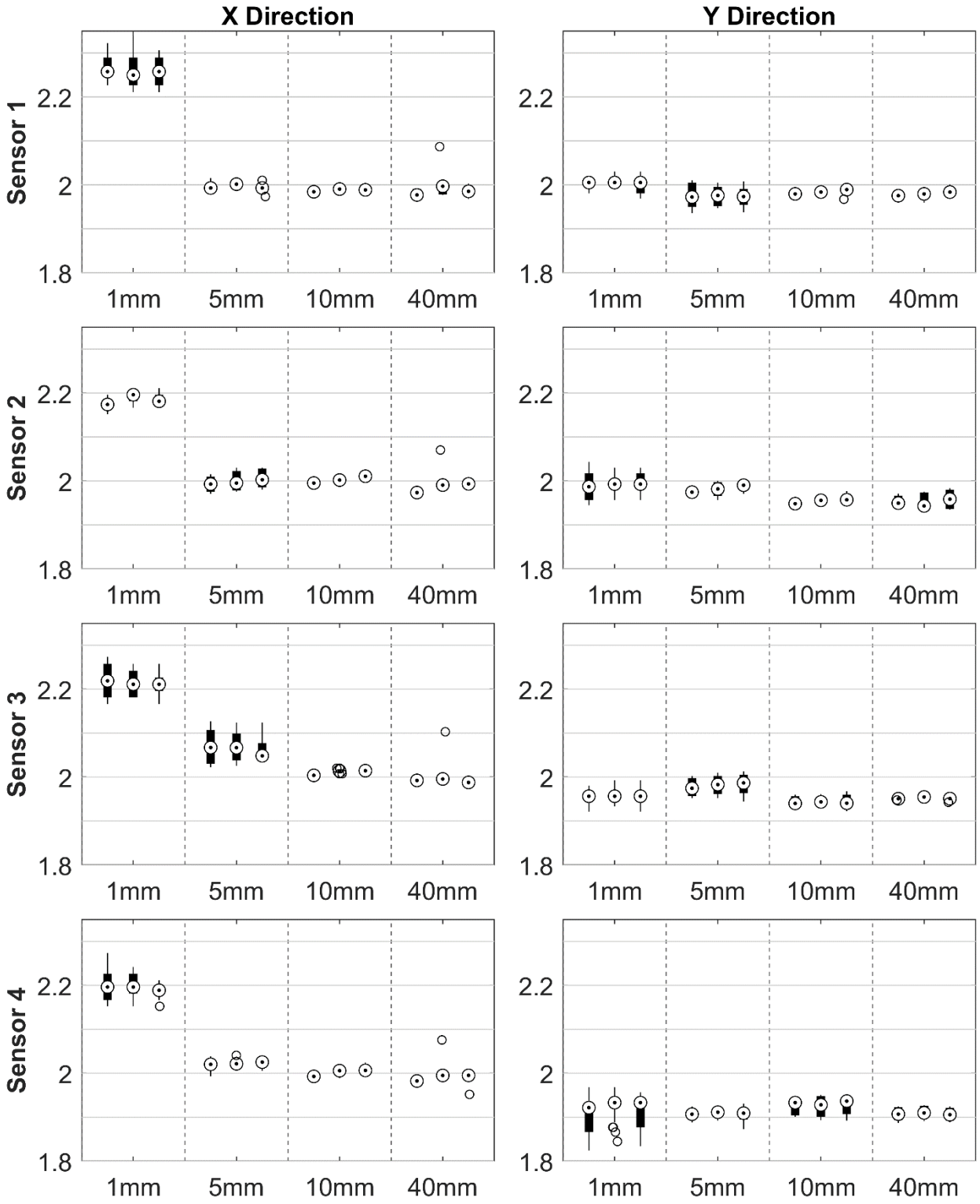

Figure A3.1: Orientation 2, square paths - Ideal calibration. Speeds are shown in ascending order within each distance grouping. 30 samples except 10 mm at 1 mm/s (24), 40 mm at 1 mm/s (10), and 40 mm at 10 mm/s (28).

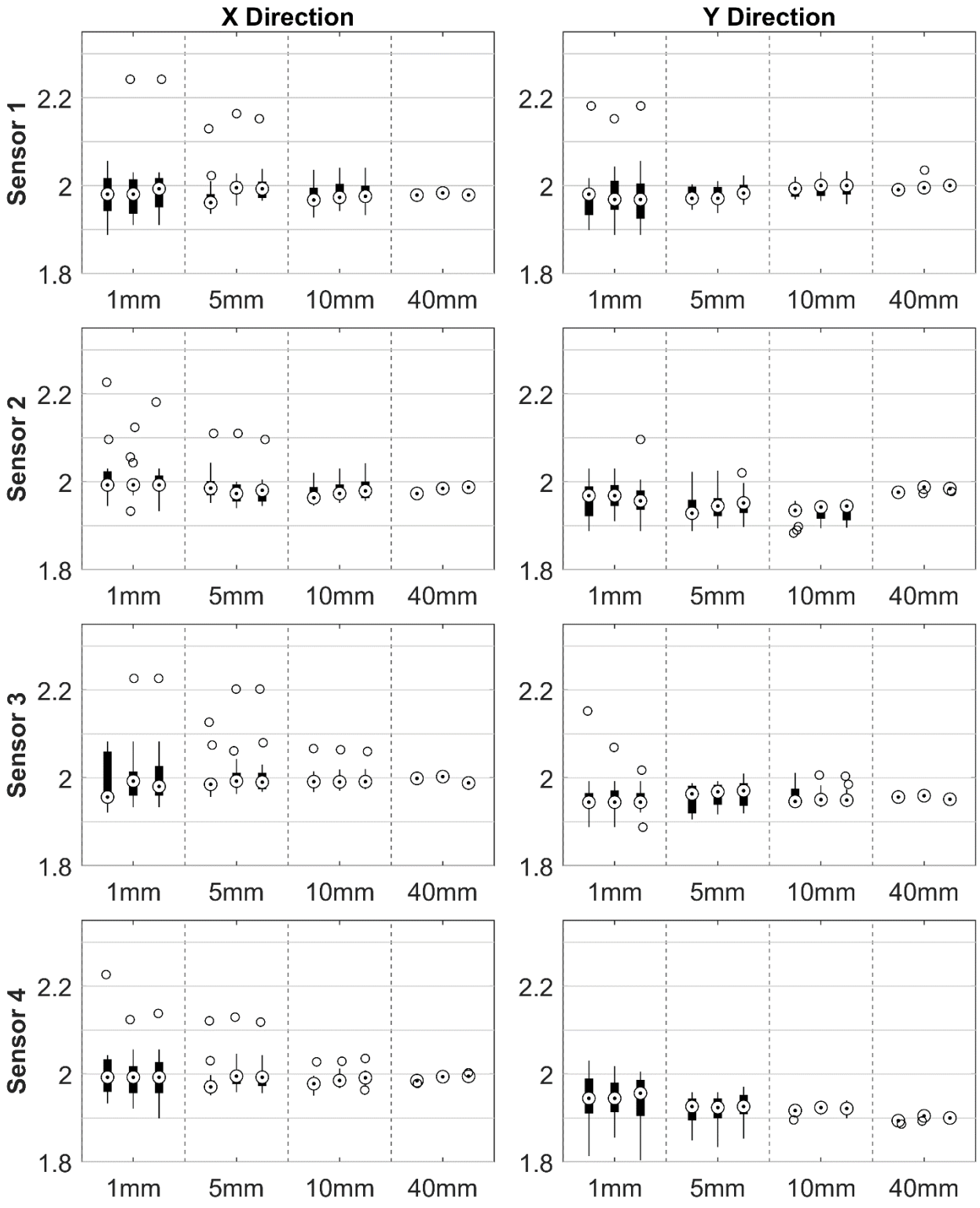

Figure A3.2: Orientation 2, linear paths - ideal calibration factors. Speeds are shown in ascending order within each distance grouping. 15 samples each except Y direction 40 mm at 1 mm/s (6) and 40 mm at 100 mm/s (14).

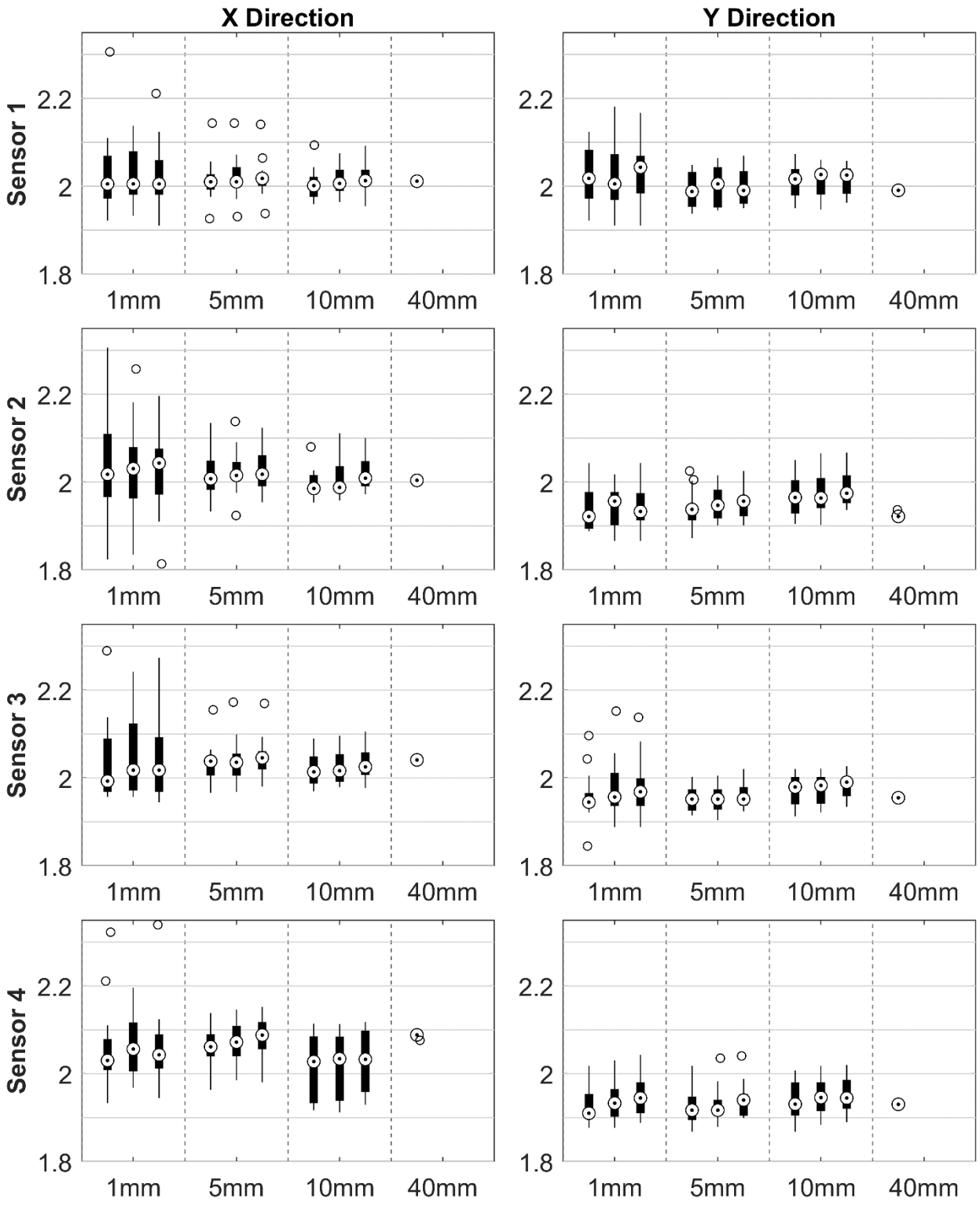

Figure A3.3: Orientation 2, diagonal paths - ideal calibration factors. Speeds are shown in ascending order within each distance grouping. 15 samples each except 40 mm at 1 mm/s (5) in both directions.

A5. Ideal Calibration Factors for both Directions

Table A5.1: Orientation 1 - Ideal calibration factors averaged over distance and speed, mean and standard deviations.

Path	Direction	Sensor 1	Sensor 2	Sensor 3	Sensor 4
Square	X	2.0003 ± 0.0109	1.9890 ± 0.0188	1.9861 ± 0.0222	1.9706 ± 0.0295
	Y	2.0082 ± 0.0234	1.9551 ± 0.0343	1.9647 ± 0.0264	1.9399 ± 0.0453
Linear	X	2.0003 ± 0.0072	1.9822 ± 0.0070	1.9880 ± 0.0076	1.9629 ± 0.0060
	Y	2.0161 ± 0.0092	1.9544 ± 0.0084	1.9609 ± 0.0163	1.9293 ± 0.0150
Diagonal	X	2.0159 ± 0.0109	1.9786 ± 0.0098	1.9919 ± 0.0060	1.9593 ± 0.0073
	Y	2.0543 ± 0.0100	2.0372 ± 0.0131	2.0362 ± 0.0088	2.0100 ± 0.0063

Table A5.2: Orientation 2 - Ideal calibration factors averaged over distance and speed, mean and standard deviations.

Path	Direction	Sensor 1	Sensor 2	Sensor 3	Sensor 4
Square	X	1.9945 ± 0.0463	1.9911 ± 0.0277	1.9952 ± 0.0349	1.9978 ± 0.0349
	Y	1.9814 ± 0.0140	1.9556 ±0.0315	1.9519 ± 0.0106	1.9084 ± 0.0340
Linear	X	1.9775 ± 0.0108	1.9839 ± 0.0061	1.9960 ± 0.0075	1.9934 ± 0.0050
	Y	2.1453 ± 0.4724	2.1316 ± 0.4721	2.0964 ± 0.4465	2.0416 ± 0.4507
Diagonal	X	2.0099 ± 0.0054	2.0046 ± 0.0057	2.0400 ± 0.0040	2.0854 ± 0.0070
	Y	1.9910 ± 0.0054	1.9240 ± 0.0071	1.9552 ± 0.0082	1.9310 ± 0.0119

A6. Plots of the Relative Error for Orientation 2

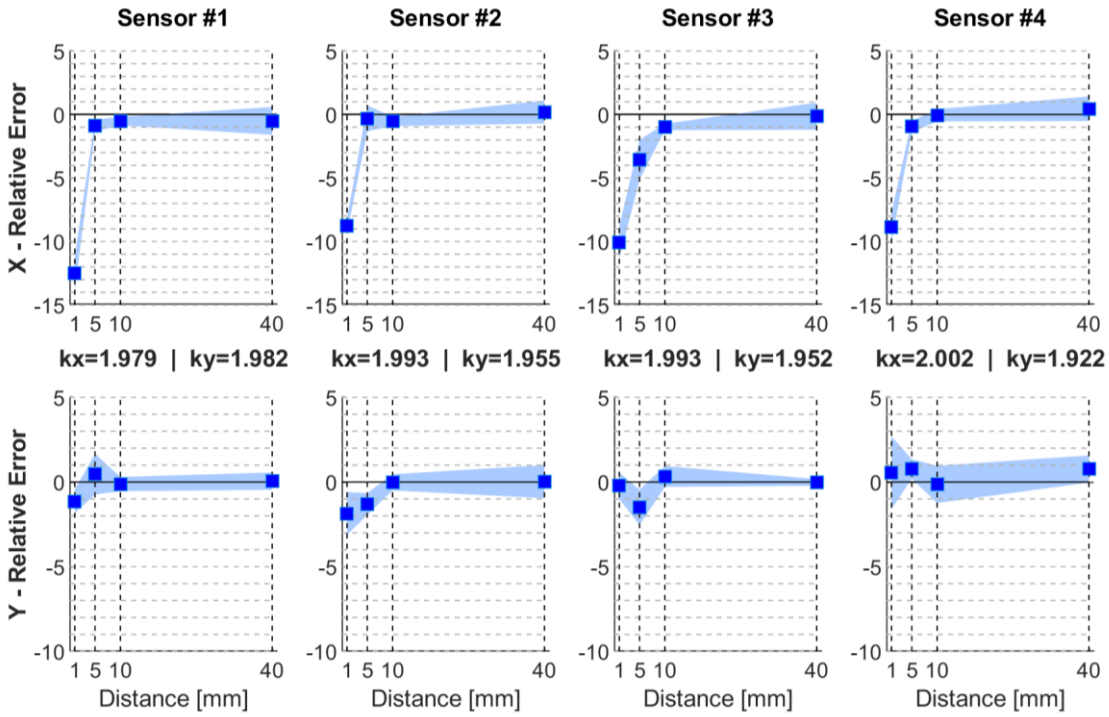

Figure A6.1: Orientation 2, square paths – relative error (speeds combined) in %. Mean (markers) and standard deviation (shaded areas), with application of speed-averaged 40 mm regional calibration factors.

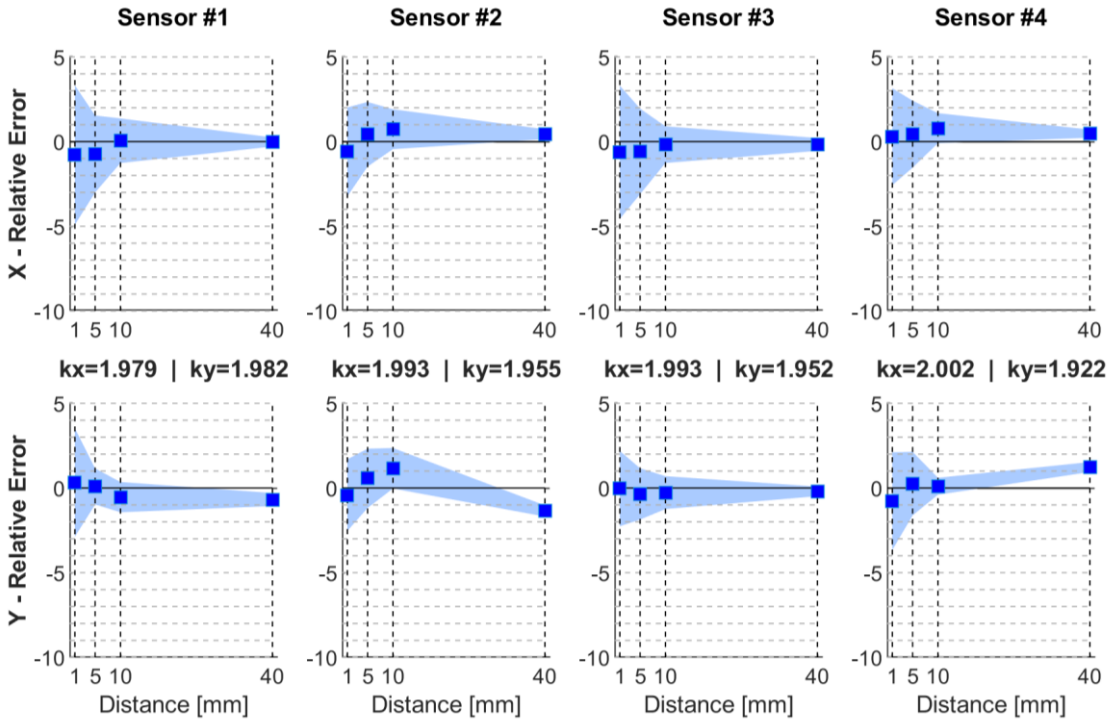

Figure A6.2: Orientation 2, linear paths – relative error (speeds combined) in %. Mean (markers) and standard deviation (shaded areas), with application of speed-averaged 40 mm regional calibration factors.

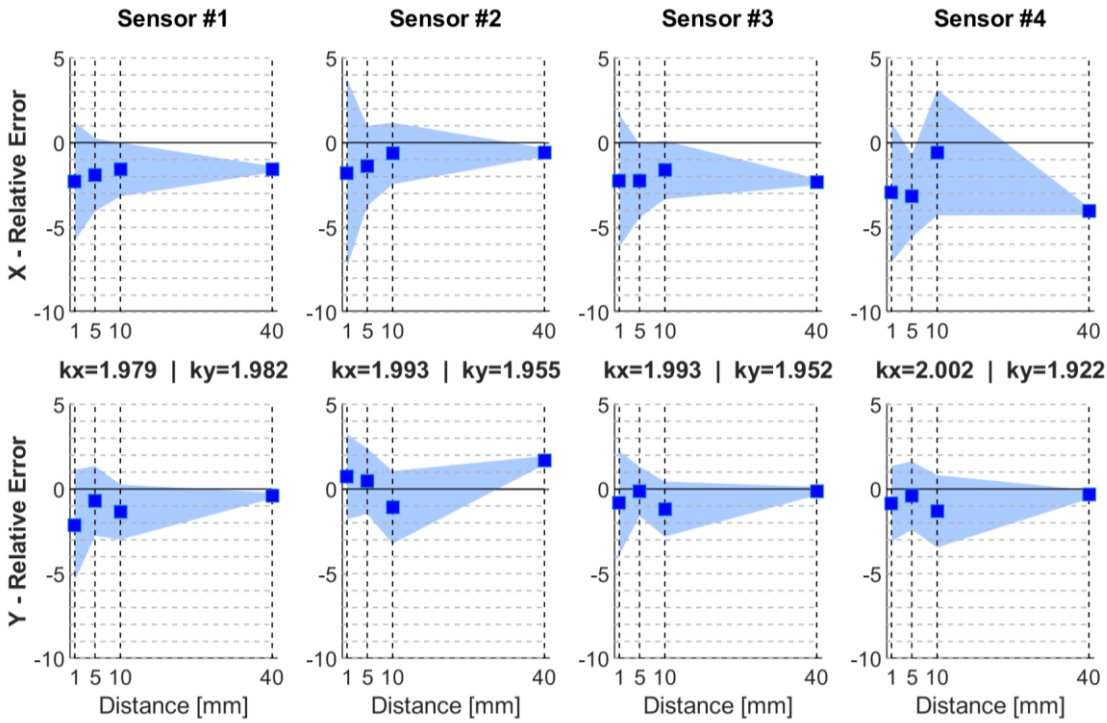

Figure A6.3: Orientation 2, diagonal paths – relative errors (speeds combined) in %. Mean (markers) and standard deviation (shaded areas), with application of speed-averaged 40 mm regional calibration factors.

A7. Maximum Means and Standard Deviations for Orientation 2

Table A7.1: Orientation 2 - maximum means and standard deviations of the relative error at 1 mm and 40 mm; number of sensor where maximum occurred in parentheses.

		Maximum at 1 mm		Maximum at 40 mm	
		Mean	Standard Deviation	Mean	Standard Deviation
Squares	**X**	-12.5019 (1)	1.3589 (1)	-0.5063 (1)	1.0939 (1)
	Y	1.8595 (2)	2.1453 (4)	0.7681 (4)	2.1453 (4)
Linear	**X**	-0.7924 (1)	4.1173 (1)	0.4670 (4)	0.3792 (3)
	Y	-0.7773 (4)	3.268 (1)	-1.3566 (2)	3.268 (1)
Diagonal	**X**	-2.9334 (4)	5.5645 (2)	-4.0484 (4)	0.2536 (2)
	Y	-2.1295 (1)	3.2712 (1)	1.6927 (2)	3.2712 (1)

A8. Plots of the Relative Drift for Orientation 2

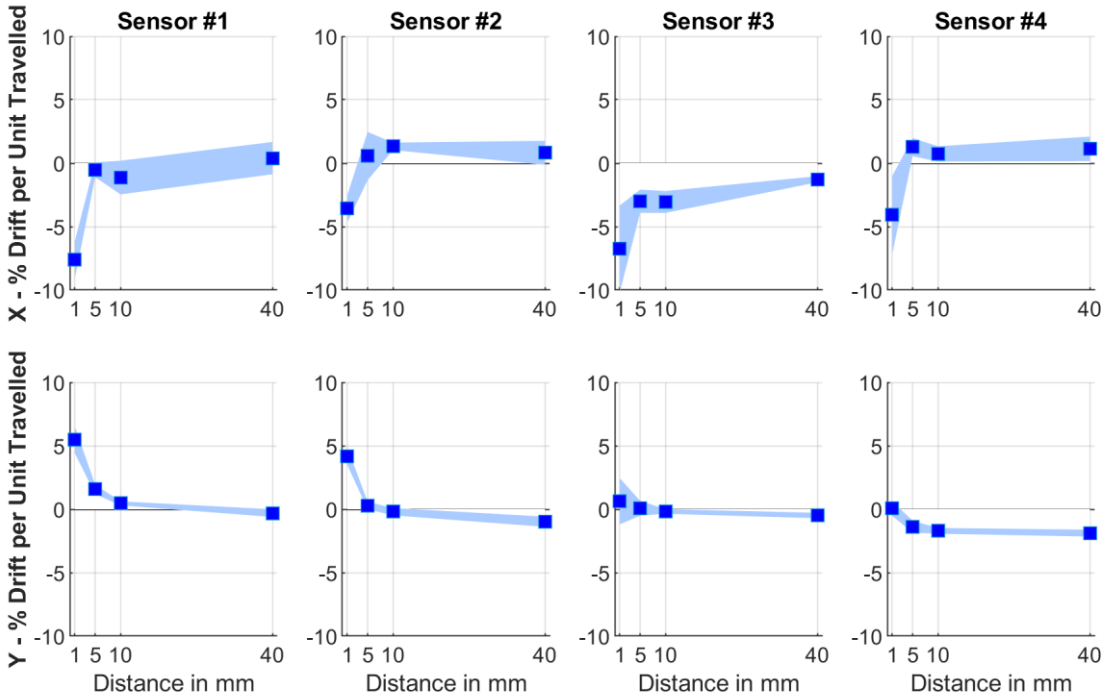

Figure A8.1: Orientation 2, square paths – mean and standard deviation of percent drift of stationary axis (X in first row, Y in second) relative to the moving axis.

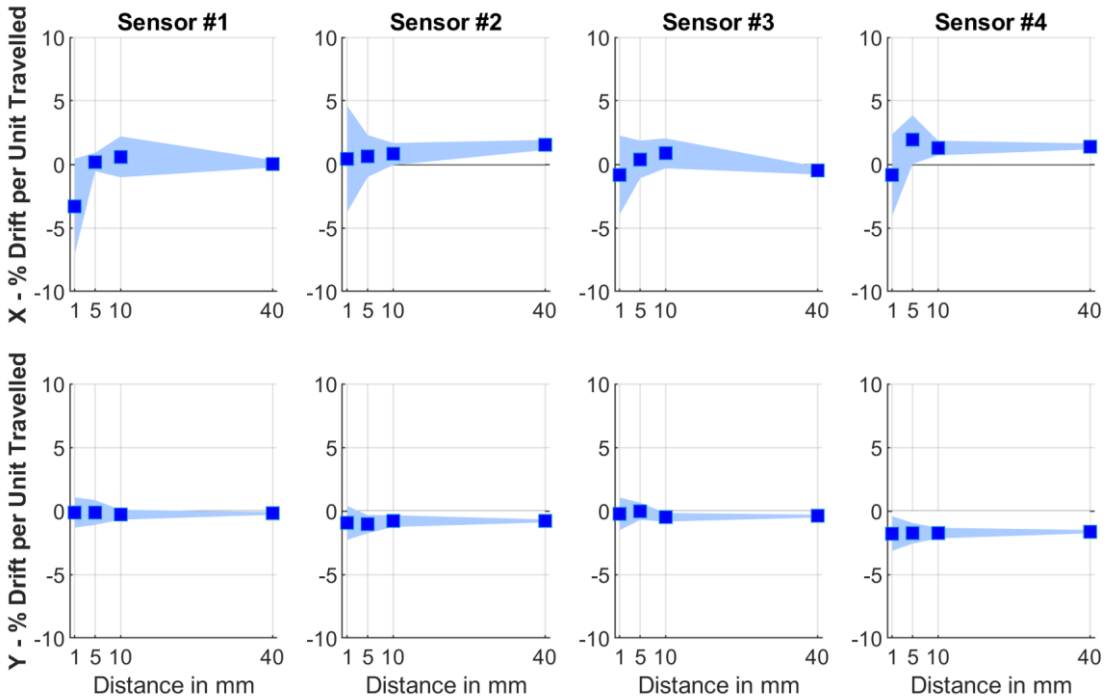

Figure A8.2: Orientation 2, linear paths - mean and standard deviation of percent drift of stationary axis (X in first row, Y in second) relative to the moving axis.

Digital Appendices

1. PDF of this document
2. Original software
3. Final software package, including user guide
4. Wilcoxon Rank Sum test p-values
5. Measurement data
6. Test Rig Kreisformtests
7. CAD Files/Technical Drawings

References

[1] Stiftung MyHandicap gemeinnützige GmbH. “Amputation und Prothese.”. URL https://www.myhandicap.de/amputation-prothese/.

[2] Center for Orthotic & Prosthetic Care. “Amputation Statistics: Fact Sheet.”. URL http://www.centeropcare.com/Portals/COPC/Amputation%20Statistics.pdf.

[3] Statistic Brain. “Amputee Statistics.”. URL http://www.statisticbrain.com/amputee-statistics/.

[4] Advanced Amputee Solutions LLC. “Amputee Statistics You Ought To Know.”, 2012. URL http://www.advancedamputees.com/amputee-statistics-you-ought-to-know.

[5] Comprehensive Prosthetics & Orthotics. “Lower Extremity.”. URL http://www.cpousa.com/prosthetics/lower-extremity/.

[6] Brigham and Women's Hospital | Department of Rehabilitation Services. “Standard of Care: Lower Extremity Amputation.”, 2011. URL http://www.brighamandwomens.org/Patients_Visitors/pcs/RehabilitationServices/Physical-Therapy-Standards-of-Care-and-Protocols/General%20-%20LE%20Amputation.pdf.

[7] University of Michigan | Michigan Medicine. “Prostheses (Lower Extremities).”. URL http://www.uofmhealth.org/conditions-treatments/lower-extremity-prostheses.

[8] Perry, Jacqueline and Burnfield, Judith M. *Gait Analysis: Normal and Pathological Function*. Slack, New Jersey (2010).

[9] Schmalz, Thomas, Blumentritt, Siegmar, and Jarasch, Rolf. “Energy expenditure and biomechanical characteristics of lower limb amputee gait: The influence of prosthetic alignment and different prosthetic components.” *Gait and Posture* Vol. 16 (2002): pp. 255–263.

[10] Kohler, Friedbert, Cieza, Alarcos, Stucki, Gerold, Geertzen, Jan, Burger, Helena, Dillon, Michael P., Schiappacasse, Carolina, Esquenazi, Alberto, Kistenberg, Robert Steven, and Kostanjsek, Nenad. “Developing Core Sets for persons following amputation based on the International Classification of Functioning, Disability and Health as a way to specify functioning.” *Prosthetics and orthotics international* Vol. 33 No. 2 (2009): pp. 117–129. DOI 10.1080/03093640802652029.

[11] Fergason, John and Smith, Douglas G. “Socket Considerations for the Patient With a Transtibial Amputation.” *Clinical Orthopaedics and Related Research* Vol. 361 (1999): pp. 76–84. DOI 10.1097/00003086-199904000-00011.

[12] Papaioannou, George, Mitrogiannis, Christos, Nianios, George, and Fiedler, Goeran. “Assessment of Internal and External Prosthesis Kinematics during Strenuous Activities Using Dynamic Roentgen Stereophotogrammetric Analysis.” *JPO Journal of Prosthetics and Orthotics* Vol. 22 No. 2 (2010): pp. 91–105. DOI 10.1097/JPO.0b013e3181cca7bb.

[13] Colombo, Giorgio, Filippi, Stefano, Rizzi, Caterina, and Rotini, Federico. “A new design paradigm for the development of custom-fit soft sockets for lower limb prostheses.” *Computers in Industry* Vol. 61 No. 6 (2010): pp. 513–523. DOI 10.1016/j.compind.2010.03.008.

[14] Cugini, U., Bertetti, M., Bonacini, D., Colombo, G., Corradini, C., and Magrassi, G. “Innovative Implementation in Socket Design: Digital Models to Customize the Product.” *Proceedings ArtA-bilitation* (2006).

[15] Grevsten, Sven and Erikson, Uno. “A Roentgenological Study of the Stump—Socket Contact and Skeletal Displacement in the PTB-Suction Prosthesis.” *Upsala Journal of Medical Sciences* Vol. 80 No. 1 (1975): pp. 49–57. DOI 10.3109/03009737509178991.

[16] Mak, Arthur F. T., Zhang, Ming, and Boone, David. “Stare-of-the-art research in lower-limb prosthetic biomechanics-socket interface.” *Journal of Rehabilitation Research and Development* Vol. 28 No. 2 (2001).

[17] Gholizadeh, Hossein, Abu Osman, Noor Azuan, Kamyab, Mojtaba, Eshraghi, Arezoo, Lúvíksdóttir, Asa Gulaug, and Wan Abas, Wan Abu Bakar. “Clinical evaluation of two prosthetic suspension systems in a bilateral transtibial amputee.” *American journal of physical medicine & rehabilitation* Vol. 91 No. 10 (2012): pp. 894–898. DOI 10.1097/PHM.0b013e31823c74d7.

[18] Noll, Veronika, Weber, Paul, Scortecci, Stephanie, Beckerle, Philipp, and Rinderknecht, Stephan. “A Sensor to Acquire the Relative Movement between Residual Limb and Prosthetic Socket.” (2016).

[19] Bocobo, Christian R., Castellote, Juan M., MacKinnon, Dougal, and Gabrielle-Bergman, Ann. “Videofluoroscopic evaluation of prosthetic fit and residual limbs following transtibial amputation.” *Journal of Rehabilitation Research and Development* Vol. 35 No. 1 (1998): pp. 6–13.

[20] Newton, Richard L., Morgan, Dan, and Schreiber, Melvyn H. “Radiological evaluation of prosthetic fit in below-the-knee amputees.” *Skeletal Radiol* Vol. 17 No. 4 (1988): pp. 276–280. DOI 10.1007/BF00401811.

[21] Eshraghi, Arezoo, Osman, Noor Azuan Abu, Gholizadeh, Hossein, Karimi, Mohammad, and Ali, Sadeeq. “Pistoning assessment in lower limb prosthetic sockets.” *Prosthetics and orthotics international* Vol. 36 No. 1 (2012): pp. 15–24. DOI 10.1177/0309364611431625.

[22] Convery, P. and Murray, K. D. “Ultrasound study of the motion of the residual femur within a trans-femoral socket during gait.” *Prosthetics and orthotics international* Vol. 24 No. 3 (2000): pp. 226–232. DOI 10.1080/03093640008726552.

[23] Convery, P. and Murray, K. D. “Ultrasound study of the motion of the residual femur within a trans-femoral socket during daily living activities other than gait.” *Prosthetics and orthotics international* Vol. 25 No. 3 (2001): pp. 220–227. DOI 10.1080/03093640108726605.

[24] Childers, W. Lee, Perell-Gerson, Karen L., and Gregor, Robert J. “Measurement of Motion Between the Residual Limb and the Prosthetic Socket During Cycling.” *JPO Journal of Prosthetics and Orthotics* Vol. 24 No. 1 (2012): pp. 19–24. DOI 10.1097/JPO.0b013e31824362e5.

[25] Board, W. J., Street, G. M., and Caspers, C. “A comparison of trans-tibial amputee suction and vacuum socket conditions.” *Prosthetics and orthotics international* Vol. 25 No. 3 (2001): pp. 202–209. DOI 10.1080/03093640108726603.

[26] von Friesen, Daniel, Grabosch, Verena, Weber, Paul, and Wendel, Kristin. “Konzeptentwicklung für ein Messsystem zur Erfassung der Relativbewegung zwischen Beinstumpf und Prothesenschaft: Concept Development of a Measuring Device for Capturing Relative Movement between

Stump and Socket in Lower Limb Prostheses.” Advanced Design Project. Technische Universität Darmstadt.

[27] Somogyi, János. “Systematische Evaluierung eines Funktionsmusters zur Erfassung der Relativbewegung zwischen Beinstumpf und Prothesenschaft.” Bachelor Thesis. Technische Universität Darmstadt.

[28] Fuchs, Christina, Robrecht, Robin, Schneider, Michael, Steidel, Verena, and Zierau, Vahid. “Evaluierung eines Funktionsmusters zur Erfassung der Relativbewegung zwischen Beinstumpf und Prothesenschaft in Gangversuchen.” Advanced Research Project. Technische Universität Darmstadt.

[29] Arnemann, Lars Christopher. “Experimentelle Evaluiering eines Funktionsmusters zur Messung von Relativbewegungen zwischen Beinstumpf und Prothesesschaft: Experimental Evaluation of a Functional Model for Recording Relative Movement between Residual Limb and Prosthetic Socket.” Bachelor's Thesis. Technische Universität Darmstadt.

[30] Levine, David, Richards, Jim, and Whittle, Michael W. *Whittle's Gait Analysis - e-book*. Elsevier (2012).

[31] CFCF. “File:Anatomical Planes.svg.”, Licensed under Creative Commons, 2014. URL https://upload.wikimedia.org/wikipedia/commons/thumb/3/31/Anatomical_Planes.svg/1109px-Anatomical_Planes.svg.png.

[32] Deluzio, Kevin J., Wyss, Urs P., Zee, Benny, Costigan, Patrick A., and Sorbie, Charles. “Principal component models of knee kinematics and kinetics: Normal vs. pathological gait patterns.” *Human Movement Science* Vol. 16 (1997): pp. 201–217.

[33] ClinicalGate. “Chapter 14: Assessment of Gait.”. URL https://clinicalgate.com/assessment-of-gait/.

[34] Noll, V., Wojtusch, J., Schuy, J., Grimmer, M., Beckerle, P., and Rinderknecht, S. “Measurement of Biomechanical Interactions at Stump-Socket Interface in Lower Limb Prostheses.” *Proceedings of the International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (2015).

[35] White, Scott C., Yack, H. John, Tucker, Carole A., and Lin, Hsing-Ying. “Comparison of vertical ground reaction forces during overground and treadmill walking.” *Medicine & Science in Sports & Excercise* Vol. 30 No. 10 (1998): pp. 1537–1542.

[36] Ottobock. “Amputationhöhe.”. URL https://www.ottobock.de/prothetik/informationen-fuer-amputierte/von-amputation-bis-rehabilitation/amputationshoehe/.

[37] Wirta, Roy W. “Analysis of below-knee suspension systems: Effect on gait.” *Journal of Rehabilitation Research and Development* Vol. 27 No. 4 (1990): pp. 385–396.

[38] Yiğter, K., Sener, G., and Bayar, K. “Comparison of the effects of patellar tendon bearing and total surface bearing sockets on prosthetic fitting and rehabilitation.” *Prosthetics and orthotics international* Vol. 26 No. 3 (2002): pp. 206–212. DOI 10.1080/03093640208726649.

[39] Du Plessis Orthotics & Prosthetics. “Lower Extremity Prosthesis.”. URL http://dportho.co.za/services/limb-prosthesis/.

[40] Ottobock. “Keeping your leg on (suspension).”. URL http://www.ottobockus.com/prosthetics/info-for-new-amputees/prosthetics-101/keeping-you-leg-on-(suspension).

[41] Ottobock. “How liners work.”. URL http://www.ottobockus.com/prosthetics/info-for-new-amputees/prosthetics-101/how-liners-work/.

[42] Söderberg, Bengt, Ryd, Leif, and Persson, Björn M. “Roentgen Stereophotogrammetric Analysis of Motion between the Bone and the Socket in a Transtibial Amputation Prosthesis.” *JPO Journal of Prosthetics and Orthotics* Vol. 15 No. 3 (2003): pp. 95–99. DOI 10.1097/00008526-200307000-00008.

[43] Lilja, M., Johansson, T., and Oberg, T. “Movement of the tibial end in a PTB prosthesis socket: a sagittal X-ray study of the PTB prosthesis.” *Prosthetics and orthotics international* Vol. 17 No. 1 (1993): pp. 21–26. DOI 10.3109/03093649309164351.

[44] Tanner, Jason E. and Berke, Gary M. “Radiographic Comparison of Vertical Tibial Translation Using Two Types of Suspensions on a Transtibial Prosthesis: A Case Study.” *JPO Journal of Prosthetics and Orthotics* Vol. 13 No. 1 (2001): pp. 14–16. DOI 10.1097/00008526-200103000-00012.

[45] Sanders, Joan E., Karchin, Ari, Fergason, John R., and Sorenson, Elizabeth A. “A noncontact sensor for measurement of distal residual-limb position during walking.” *The Journal of Rehabilitation Research and Development* Vol. 43 No. 4 (2006): p. 509. DOI 10.1682/JRRD.2004.11.0143.

[46] Commean, Paul K., Smith, Kirk E., and Vannier, Michael W. “Lower extremity residual limb slippage within the prosthesis.” *Arch Phys Med Rehabil* Vol. 78 (1997): pp. 476–485.

[47] Gholizadeh, Hossein, Abu Osman, Noor Azuan, Lúvíksdóttir, Ása Gulaug, Eshraghi, Arezoo, Kamyab, Mojtaba, and Wan Abas, Wan Abu Bakar. “A new approach for the pistoning measurement in transtibial prosthesis.” *Prosthetics and orthotics international* Vol. 35 No. 4 (2011): pp. 360–364. DOI 10.1177/0309364611423130.

[48] Narita, H., Yokugushi, K., Shii, S., Kakizawaa, M., and Nosaka, T. “Suspension effect and dynamic evaluation of the total surface bearing trans-tibial prosthesis - a comparison with the patellar tendon bearing trans-tibial prosthesis.” *Prosthetics and orthotics international* Vol. 21 (1997): pp. 175–178.

[49] Ng, T. W. “The optical mouse as a two-dimensional displacement sensor.” *Sensors and Actuators A: Physical* Vol. 107 No. 1 (2003): pp. 21–25. DOI 10.1016/S0924-4247(03)00256-5.

[50] Minoni, Umberto and Signorini, Andrea. “Low-cost optical motion sensors: An experimental characterization.” *Sensors and Actuators A: Physical* Vol. 128 No. 2 (2006): pp. 402–408. DOI 10.1016/j.sna.2006.01.034.

[51] Agilent Technologies. “Agilent ADNS-2051 Optical Mouse Sensor - Data Sheet.” (6 May 2003).

[52] Avago Technologies. “ADNS-9800 LasterStream™ Gaming Sensor - Data Sheet.”, Data Sheet (13 January 2012).

[53] Palacin, J., Valgañon, I., and Pernia, R. “The optical mouse for indoor mobile robot odometry measurement.” *Sensors and Actuators A: Physical* Vol. 126 No. 1 (2006): pp. 141–147. DOI 10.1016/j.sna.2005.09.015.

[54] SparkFun. “Serial Peripheral Interface (SPI).”. URL https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi.

[55] National Instruments. “NI myRIO-1900: User Guide and Specifications.” (May 2016).

[56] Arduino. “Arduino Uno Rev3 - Tech Specs & Documentation.”. URL https://store.arduino.cc/arduino-uno-rev3.

[57] Avago Technologies. “ADNS-6190-002 LasterStream™ Gaming Round Lens - Data Sheet.” (16 August 2011).

[58] Tindie Online Shop. “ADNS-9800 Laser Motion Sensor.”. URL https://www.tindie.com/products/jkicklighter/adns-9800-laser-motion-sensor/.

[59] mrjohnk. “Avago ADNS-9800 Test (Sample Arduino Sketch).”, 2016. URL https://github.com/mrjohnk/ADNS-9800.

[60] Arduino. “Arduino Mega 2560 REV3 - Tech Specs.”. URL https://store.arduino.cc/arduino-mega-2560-rev3.

[61] Arduino. “Arduino Due - Tech Specs & Documentation.”. URL https://store.arduino.cc/arduino-due.

[62] Arduino. “Genuino Zero - Tech Specs.”. URL https://store.arduino.cc/genuino-zero.

[63] Arduino. “Arduino MKR1000 WIFI - Tech Specs.”. URL https://store.arduino.cc/arduino-mkr1000.

[64] Arduino Forum. “Port manipulation PINx commands with Arduino Due?”, 2012. URL http://forum.arduino.cc/index.php?topic=129868.15.

[65] graynomad. “The Unofficial Arduino Due Pinout Diagram.”, 2014. URL http://www.robgray.com/temp/Due-pinout.pdf.

[66] tutorialspoint. “C- Unions.”. URL https://www.tutorialspoint.com/cprogramming/c_unions.htm.

[67] MathWorks. “Matlab Documentation - drawnow.”. URL https://de.mathworks.com/help/matlab/ref/drawnow.html?searchHighlight=drawnow&s_tid=doc_srchtitle.

[68] SparkFun. “Serial Terminal Basics.”. URL https://learn.sparkfun.com/tutorials/terminal-basics/serial-terminal-overview-.

[69] SourceForge. “Realterm: Serial Terminal - Documentation.”. URL https://realterm.sourceforge.io/.

[70] SourceForge. “Serial Terminal: Controlling RealTerm from Matlab.”. URL https://realterm.sourceforge.io/realterm_from_matlab.html.

[71] Digi International, Inc. “Digi XBee3 802.15.4.”. URL https://www.digi.com/products/xbee-rf-solutions/2-4-ghz-modules/xbee3-802-15-4#specifications.