AXIAL MOTION COMPENSATION OF OPTICAL COHERENCE TOMOGRAPHY WITH A COLLABORATIVE MEDICAL ROBOT

by

Robnier Reyes Perez

B.Eng., Ryerson University, Toronto, Ontario, 2016

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2020

Abstract

AXIAL MOTION COMPENSATION OF OPTICAL COHERENCE TOMOGRAPHY WITH A COLLABORATIVE MEDICAL ROBOT

Robnier Reyes Perez

Master of Applied Science, Electrical and Computer Engineering

Ryerson University, 2020

This thesis presents an imaging tool consisting of an Optical Coherence Tomography (OCT) imaging system mounted on a collaborative robotic arm to enable axial motion compensation. Optical Coherence Tomography is a subsurface, high-resolution imaging modality used in neuroimaging to differentiate between pathological and non-pathological tissue. The motivation behind this project is to bring Optical Coherence Tomography to the operating room for neuroimaging to help with cancerous tissue differentiation and maximize the extent of tumor resection. However, neurosurgeons have expressed concern with respect to intracranial pressure (ICP) pulsation displacing the brain far off the optic axis of the imaging system so as to not be visible. The collaborative robotic arm compensates for sample motion along the optic axis using a Proportional controller to track the position of the peak intensity of the sample's intensity profile, which generally corresponds to the sample surface. Collaborative robots have changed the robot industry paradigm becoming increasingly functional and safer than the previous generations of robotic arms. We present an OCT robot end-effector to test the feasibility of performing OCT imaging with the collaborative robot.

Acknowledgements

I'd like to thank my supervisor, Dr. Victor X.D. Yang, for guiding me through the steps to complete my thesis. I'd like to thank my friend and colleague, Andrew Marques, for his moral support. I'd like to thank the rest of my lab group for their constant motivation.

I'd like to thank my wife, Sarah, and my family for always being there.

Table of Contents

List of Figures

List of Equations

This page was left blank on purpose.

1.Introduction

1.1 Background and Motivation

Cancer is a leading cause of death and brain cancer is one of the most common and deadly types. Despite advances in modern surgical techniques, there is a limit to the extent of tumor resection current surgeons can achieve because visual delimitation of the tumor’s margin is still a challenge. Optimal safe tumor resection is of upmost importance because the rate of cancer recurrence is proportionally correlated to the extent of the resection. Therefore, novel approaches to allow for intraoperative differentiation between cancer and non-cancer tissue are needed to improve clinical outcome [1]. The motivation behind this project is to develop an Optical Coherence Tomography (OCT) intraoperative imaging tool to introduce in the workflow of brain tumor resection surgical procedures. OCT is a contactless medical imaging modality useful in applications where high-resolution, subsurface images are required. The applications of OCT for neuroimaging were first presented by Boppart as a useful tool for brain tissue differentiation [2]. Bohringer has also demonstrated the different optical properties between pathological and nonpathological brain tissue by comparing OCT images with histology results and presenting light attenuation maps showing the differences in attenuation profiles of tumors and healthy tissue [3] [4]. These findings have been confirmed by Dr. Quiñones in a study where pathologically confirmed brain cancer tissue was found to have had considerably lower attenuation than non-cancerous tissue [4]. This different attenuation profiles of cancerous and noncancerous tissue can be observed under OCT as structures with different intensities. His latest research at John-Hopkins has also shown that OCT imaging can be used to distinguish brain tumor margins [5] [6].

Optical Coherence Tomography was developed as an imaging modality by Fujimoto and his research group in the early 1990's from principles of interferometry [7]. Interferometry allows one to measure the delay of reflected and scattered light from a sample by comparing it to a known path length. These principles had been used in optic fiber communication and meteorology, but the new implementation allowed images of a biological sample to be obtained and it was made popular for structural imaging of transparent (i.e. eye) and nontransparent (i.e. skin, brain) tissues by the work of Fercher and Izatt, among others [8] [9] . OCT has been traditionally referred to as optical biopsy given its use to obtain histology-like images of tissue. During research studies, histology slices of tissue are often compared to or analyzed next to their corresponding OCT images. Today, commercial OCT systems for retinal imaging are often encountered as a benchtop system in ophthalmology clinics. OCT has also become an important research tool with applications in cardiology, dermatology, and oncology, where hand-held probes are used to perform imaging in vivo. OCT is analogous to ultrasound in the sense that a wave is emitted, and the time delay and magnitude of the reflected or scattered wave is measured to obtain structural information of the sample. In contrast to ultrasound, which uses sound waves to detect differences in acoustic impedance, OCT uses light waves to detect differences in indices of refraction in the structures within the sample. These axial intensity profile produced by the backscattered waves can be used to construct cross-sectional images of the sample. Moreover, the speed of sound ($3 * 10^3 m/s$) and light speed ($3 * 10^8 m/s$) differences means that time delay and magnitude measurements for ultrasound can be done directly by the sensor, whereas in OCT these measurements is done indirectly using low-coherence interferometry. Low-coherence interferometry splits a light beam and correlates the echo time delay and backscattered intensity of the structures within the sample relative to a known reference path length. The resolution of this measurement is directly related to the temporal coherence length of the source since interference can only be achieved when the optical path difference is within the coherence length. The coherence of a laser source relates the phase relationship of the emitted waves at different locations (spatial) and times (temporal). The coherence of a laser defines its ability to create interference. In general, laser beams exhibit high spatial coherence which is what makes the beam have a high quality and directionality. On the other hand, the temporal coherence of a laser very much depends on the type of laser. Therefore, a low-coherence laser is meant to describe its temporal coherence characteristic. High-temporal coherence lasers (i.e. HeNe laser) are very monochromatic and exhibit a fixed phase relationship over a relatively long distance. A pure sinusoidal wave, Fig 1a, emitted over a finite period of time, Δt, and extending over a length, $l = c\Delta t$, is perfectly coherent because the phase of every wave emitted by the source is predictable. For a perfectly coherent wave, these are the coherent time and length of the wave. In practice, however, lasers have a spectrum of frequencies, $\Delta\nu$, at which the emitted waves oscillate. These superpose to create a wave packet, Fig 1b&c, whose coherence length depends on the relationship between the time the emission occurs, Δt, and the source spectrum, $\Delta\nu$. The wave packet is modulated by an envelope function that depends on the laser source used. If a Gaussian function is taken to be the envelope of the wave packet, Fig 1d, the coherence time of the laser is inversely proportional to its spectrum, $\Delta t = {}^{1}/_{\Delta\nu}$. Therefore, a broadband laser source (i.e. supercontinuum) is characterized by a short temporal coherence. Furthermore, the assumption of a Gaussian envelope has practical mathematical implications as the Fourier transform of a Gaussian function is also Gaussian. A Fourier transform is used to deconstruct the sinusoidal oscillations that make up the wave and observe its spectrum, Fig 1d.

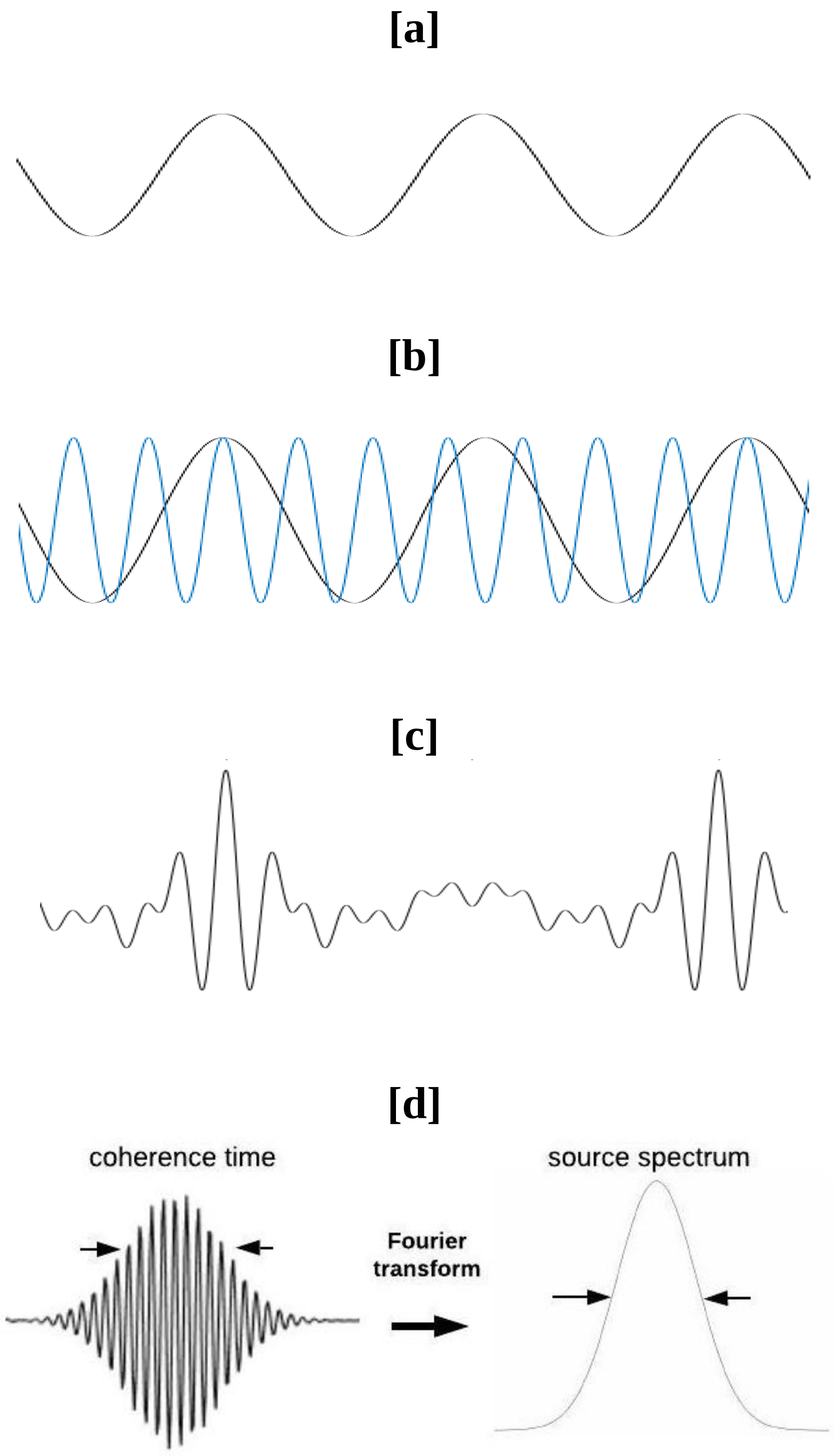


Figure 1. Temporal coherence of light. [a] Monochromatic sinusoidal wave of finite length. [b] Dual-frequency emission. [c] Beat wave. [d] Gaussian beat wave and its Fourier transform.

OCT imaging technology has evolved from when it was first proposed as a structural time-domain imaging modality to frequency-domain modalities offering structural and functional information about the sample. In the following discussion we outline time-domain and frequency-domain structural imaging schemes to introduce the reader to the requirements of building an OCT imaging system and its functionality.

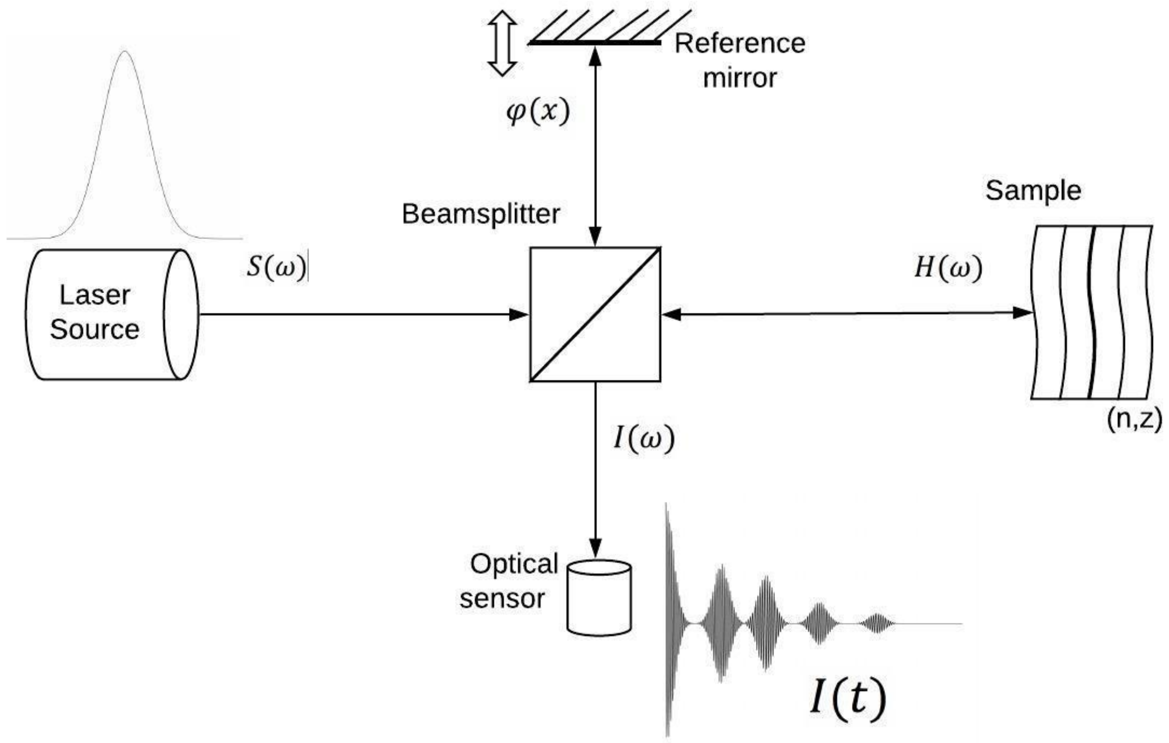


Figure 2. Schematic diagram for Time-Domain Optical Coherence Tomography (TD-OCT)

Figure 2 depicts a Michelson interferometer with a beamsplitter cube, translating mirror at the reference arm, sample under test, and optical sensor to detect the interference signal. The translating mirror at the reference arm is crucial in time-domain OCT as it allows the interferometer to match optical paths with structures deeper in tissue. Mirror translation interrogates the sample and forms the intensity depth profile, commonly referred to as an

Amplitude-line, or just A-line. We can present a mathematical formulation for OCT by expressing the laser source as an electric field:

$$E_{source}(\omega, t) = S(\omega)e^{-i\omega t}$$

Equation [1] Electric field with a Gaussian amplitude spectrum

where, $S(\omega)$ is the source field amplitude spectrum, ω is the optical frequency as it relates to wavelength through $\omega = {}^{2\pi c}/_{\lambda}$, and time variation *t*. The source field amplitude spectrum is a Gaussian envelope with a distribution defined by the frequency characteristics of the laser source:

$$S(\omega) = e^{\frac{-(\omega-\omega_c)^2}{2\sigma^2}}$$

Equation [2] Gaussian amplitude spectrum

The beamsplitter cube separates the laser beam so that the two waves travel through equal-in-length paths along the reference and sample arm, respectively. The reflection and scattering signals from each arm travel once again back to the beamsplitter where they form an interference pattern. Assuming a lossless beamsplitter with a 50:50 split ratio:

$$E_{ref}(\omega, t, \Delta x) = \frac{1}{2}\, E_{source}(\omega, t)e^{-i\varphi(\Delta x)}$$

$$E_{sample}(\omega, t) = \frac{1}{2}\, E_{source}(\omega, t)H(\omega)$$

$$E_{out}(\omega, t, \Delta x) = E_{ref}(\omega, t, \Delta x) + E_{sample}(\omega, t)$$

Equation [3] Electric fields of the interferometer arms for the time-domain OCT scheme.

where, $\varphi(\Delta x)$ is a function of phase accumulation with respect to the reference mirror displacement $\Delta x = \Delta tc/n_{air}$. The need for the interferometry comes because the time of flight for light, Δt, is too small to be directly measured by modern sensors and electronics. However, the cosine modulated phase delay accumulation between interferometer arms can be detected.

$$\varphi(\Delta x) = \cos\,(2\Delta x\, n_{air}\,\omega/c)$$

Equation [4] Phase delay caused by the reference mirror displacement.

When $\Delta x = 0$, the reference and sample arms lengths are matched and the cross-interference between interferometer arms is obtained. Reference mirror translation, Δx, is said to interrogate deeper layers of the sample and extract its structural composition as an intensity profile. The smallest structure detectable in the axial direction is of the same size as the coherence length of the source used for imaging. This means that the axial resolution of the imaging system is equal to the coherence length of the laser source.

In this thesis we use a frequency-domain OCT method with a broadband laser and a spectrometer; a general schematic diagram is depicted in Fig 2. The two relevant distinctions are the change to a fixed reference mirror and the use of a spectrometer as a detector. Spectrometers are optical devices designed to separate the wavelength components of light sources and observe their spectral density, the intensity distribution of the wavelengths.

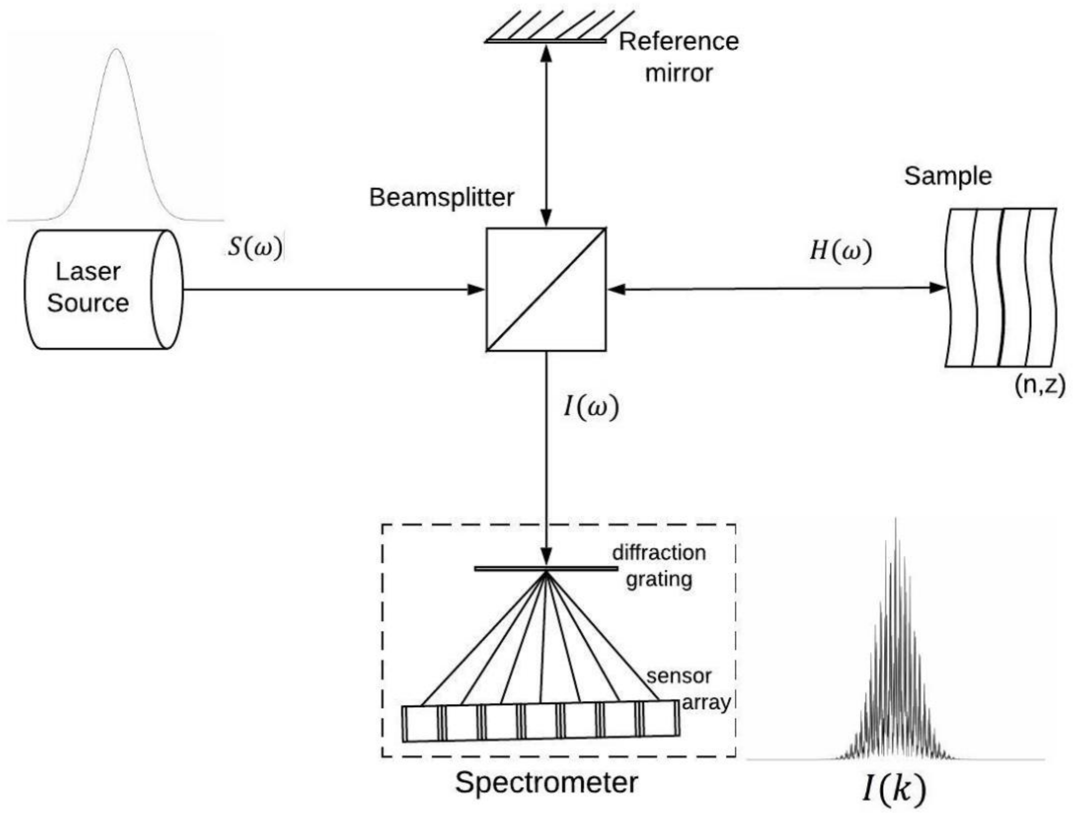


Figure 3. Schematic diagram for frequency-domain Optical Coherence Tomography.

We continue with the assumption of a laser source with Gaussian amplitude spectrum and a lossless beamsplitter with a 50:50 split ratio. However, the fixed mirror eliminates the phase accumulation from the optical delay:

$$E_{ref}(\omega,t) = \frac{1}{2}\, E_{source}(\omega,t)$$

$$E_{sample}(\omega,t) = \frac{1}{2}\, E_{source}(\omega,t)H(\omega)$$

$$E_{out}(\omega,t) = \frac{1}{2}\, E_{source}(\omega,t)(1 + H(\omega))$$

Equation [5] Electric fields of the interferometer arms for the frequency-domain OCT scheme.

where, $H(\omega)$ is the frequency-domain response function of the sample under test:

$$H(\omega) = \int_{-\infty}^{\infty} r(\omega, z) e^{-i2n(\omega,z)\omega z/c}\, dz$$

Equation [6] Sample response function in the frequency domain.

The response function contains structural information of the sample in the functions $r(\omega, z)$ and $n(\omega, z)$ as they describe the frequency and depth dependent reflectivity and group index of refraction of all the structures in the sample along the optic axis of the beam (z-axis). The electric field power intensity at the output of the beamsplitter contains the average reflectivity of the reference mirror and sample information. From the frequency-domain response function of the sample, $H(\omega)$, we can see that the desired structural information is encoded in this function. However, in order to decode this information, we must observe the spectral density of the interference signal with the use of a spectrometer. The spectrometer diffracts the beam into its wavelength components, and it detects the particular intensity of each component. Once the optical signal has been digitized in k-space, where $k = {}^{2\pi}/_{\lambda}$, a Fourier Transform operation can be performed on the signal to reconstruct the depth intensity profile of the sample:

$$I(t) = FT\{I(k)\}$$

Equation [7] Fourier transform operation

The motivation behind the thesis is to integrate novel medical technologies to create innovative imaging tools for intraoperative neuroimaging applications. The potential applications for OCT have never been lacking. However, further technology progress is needed to increase adoption. OCT imaging systems have evolved over the decades from large systems on optic tables in research labs through portable cart systems to compact benchtop systems with handheld probes. The portability and form factor of these probes have improved over time with increase application in several clinical settings [10]. A forward step in the development of OCT imaging tools is to leverage the advantages of collaborative robotic arms to hold the probe and navigate the sample. To this end, we designed and constructed an OCT contactless probe as a robot end-effector to test the feasibility of imaging with the LBR Med collaborative robot.

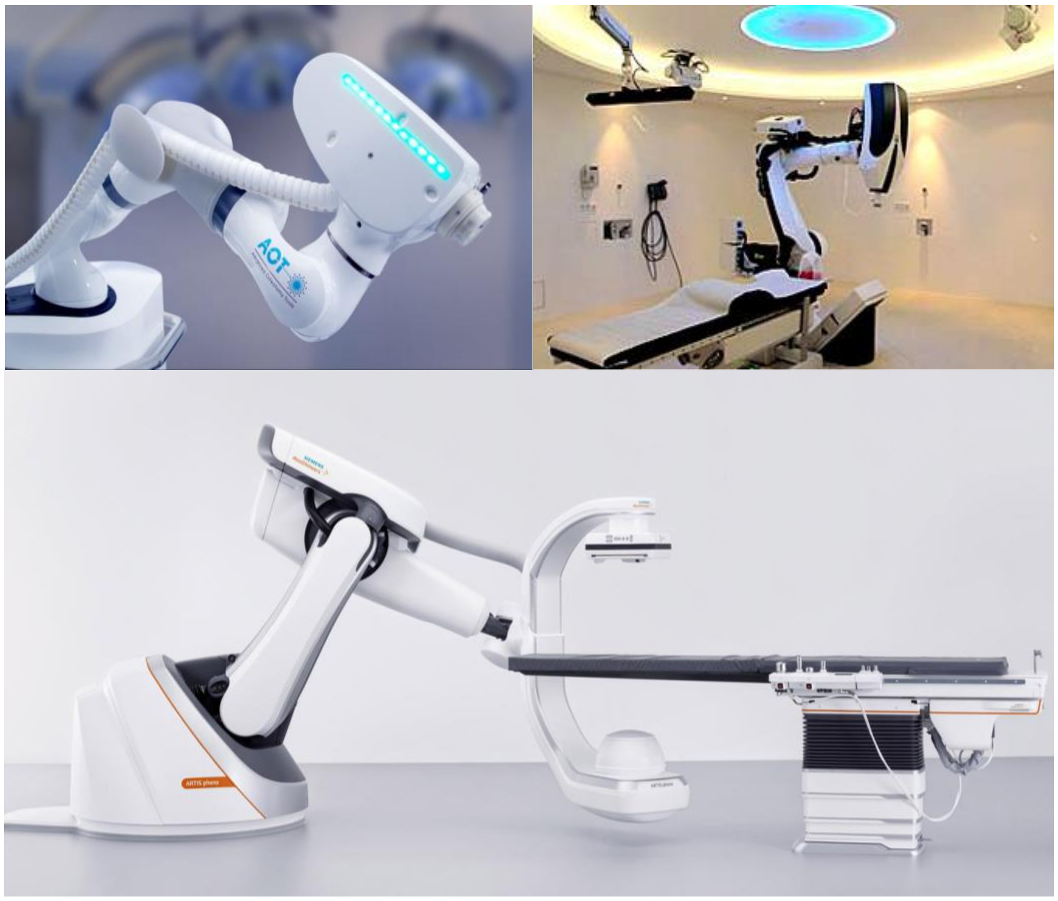


Figure 4. CARLO (top-left), Cyberknife (top-right) Artis pheno (bottom), images taken from the public domain.

Applications of collaborative robots in the medical field are becoming increasingly popular with start-ups starting to partner with industry leaders to bring commercial systems to market. Robot-assisted laser and radiation delivery systems for navigation, imaging or treatment applications offer a lower level of risk given the contactless nature of the application. A number of medical applications have been showcased by robot manufacturers (i.e. KUKA, ABB) and partners (i.e. AOT, Accuray) including laser ablation osteotomy, targeted-radiation delivery, and ultrasound imaging. As a result, the robot maintains a safe distance from the patient. The distance kept between the robot and the patient provides a margin of safety to stop the robot in case of a fault (i.e. unexpected force). This creates a safe use case for patients. Research and development of imaging robotic systems is advancing and showing good results with the development of commercial systems, Fig 4. The CARLO system is used for cranio-maxillofacial surgery and consists of a cold laser ablation payload held by an LBR Med, a 7 degrees-of freedom (DOF) robotic arm. The system is produced by Advanced Osteotomy Technologies AG, founded in 2011 as a spin-off from the University of Basel [11]. Cyberknife (Accuray Inc, USA) makes use of a KUKA Quantec robotic arm to deliver radiation and treat tumors. The system leverages the flexibility and precision of robotic arms to target tumors from different angles with high accuracy. This way they minimize the overall radiation received by healthy tissue while keeping the cancerous tumor on target [12]. Artis pheno (Siemens AG, Germany) is a commercial robotic intraoperative imaging tool used in the operating room. The system consists of a KUKA Quantec robot holding C-arm at the end-effector. The imaging versatility of such system makes it easy to adapt to different procedures [13]. Robot-assisted intraoperative OCT imaging is in its research state.

In this thesis we address the issue raised by surgeons that a contactless OCT imaging probe will suffer from image instability due to intracranial pressure (ICP) pulsation. ICP pulsation is the result of cerebrospinal fluid movement, cardiac circulation, and breathing patterns. ICP pulsation has been studied under MRI and ultrasound in order to better understand brain function. During these studies, motions with an amplitude in the submillimetre range have been observed to occur [14] [15]. However, after a craniotomy the brain is no longer enclosed by the skull and the motion is more pronounced. If OCT is to be adopted as an intraoperative imaging modality under robot control, the problem of ICP pulsation needs to be addressed. Otherwise, the image will be unstable and may drift out of the field-of-view (FOV) of the imaging system. We present a peak intensity detection solution to track the surface of the sample and a Proportional controller to position the end-effector along the optic axis (z-axis).

In the following chapters, the work done by the author will be presented. The thesis is divided into chapters corresponding to background and motivation behind the thesis, the implementation of OCT imaging and the construction of the end-effector, and consequently the integration with the collaborative robot through safety-oriented programing to achieve motion compensation. The OCT system uses a laser source, interferometer, spectrometer, and the appropriate signal acquisition and processing hardware and software. Safety-oriented programming of the robot requires a risks and hazards assessment to understand the dangers present in a surgical setting. These risks and hazards are variable in nature, defined by the specific surgical setting characteristics (i.e. staff presence, nearby equipment). An adaptive safety-oriented programming approach is taken to maximize safety as the robot's end-effector moves closer to the patient. System integration consists of designing and manufacturing a robot end-effector for the interferometer and aiming laser. Also, an ethernet-based communication channel between the PC hosting the OCT program and the robot controller is established to create the control loop for motion compensation. The proportional controller tuning as well as its frequency response are presented. Results for motion compensation of an intracranial pressure (ICP) waveform are presented.

2.Optical Coherence Tomography Imaging System

Since the inception of Optical Coherence Tomography (OCT), development of solid-state and fiber-based optical components designed for and with OCT in mind, have facilitated the construction of these systems. OCT imaging system for research applications require flexibility of control over parameters of the system (i.e. data acquisition, trigger signals) so customize systems are very common in research labs. Programmable laser and camera, high-precision galvo motor, and data acquisition and signal processing software platform such as LabVIEW are leveraged to build the OCT system. In this chapter we describe the steps taken for the implementation of the OCT imaging system and the construction of a probe as the robot’s end-effector. The system uses a supercontinuum (SC) laser source from NKT Photonics (Copenhagen, Denmark) and custom-built spectrometer from P&P Optica (Ontario, Canada). It also uses off-the-shelf solid-state and fiber-based optic components from Thorlabs to construct the interferometer. Data acquisition hardware and software from National Instruments were chosen to build customized tasks and a graphical user interface. A detailed schematic of all components used in shown in the optic diagram of Fig 5. We proceed to explain each component as it relates to the work done by the author.

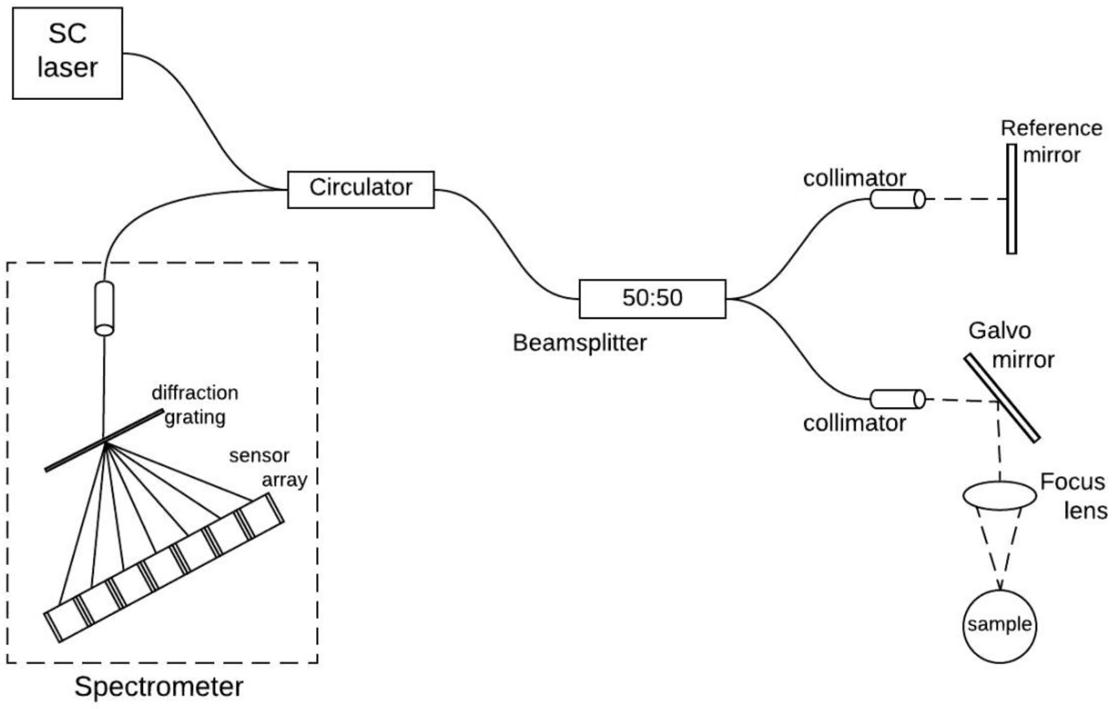


Figure 5. Optical Coherence Tomography imaging system optic diagram.

2.1 Laser Source

We have selected to use a supercontinuum (SC) laser SuperK Extreme EXR-1 from NKT Photonics (Denmark) for its broadband and high-power output. NKT Photonics was an early adopter of a fabrication process to develop a nonlinear optic fiber known as photonic crystal fiber. Photonic crystal fiber has a unique index profile, shown in Fig 6, different from that of single-mode fiber, and by arranging the honey-comb structure a variety of properties (i.e. dispersion, birefringence, nonlinearities) can be achieved.

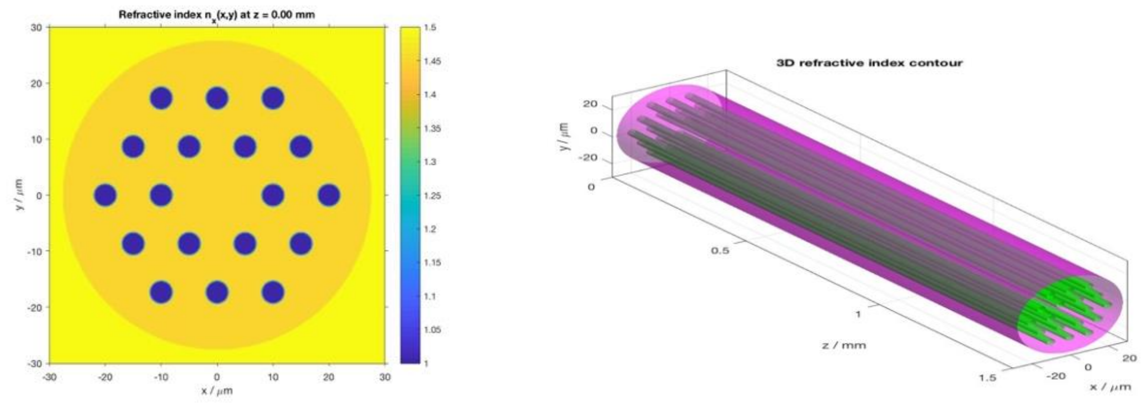

Refractive index $n_x(x,y)$ at z = 0.00 mm
x / µm
y / µm
3D refractive index contour
z / mm

Figure 6. Photonic Crystal Fiber (PCF) refractive index structure. Rendering done in MATLAB using BeamLab toolbox.

Supercontinuum (SC) lasers are made possible by the interaction between light and nonlinear optic materials. When a high-power laser pulse is propagated through a photonic crystal fiber, the interaction causes the pulse to broaden. The wide broadening of the incident laser pulse is referred to as supercontinuum generation. The broadening is cause by frequency conversion processes (i.e. frequency doubling, sum and difference frequency generation) [16]. Supercontinuum lasers are useful for OCT imaging applications. Their broadband output makes them ideal for high resolution imaging and a high-power output ensures high sensitivity [17] [18]. OCT imaging has been demonstrated in a number of different optical windows using supercontinuum lasers [19] [20] [21] [22] [23] [24]. The spectrum provided by the SuperK Extreme, 0.4-2.4 μm, is filtered to a Gaussian-like spectrum centered at 1.31 μm ($\lambda_0 = 1310nm$) with a bandwidth of $\Delta\lambda$ = 135nm, using the SuperK Gauss, as seen in Fig 7. The filtered beam is coupled into a single-mode fiber delivery system, SuperK Connect, and terminated with a FC/APC connector for easy coupling with the fiber-based interferometer. The power output of the laser is set to its minimum setting of 82mW for safety reasons.

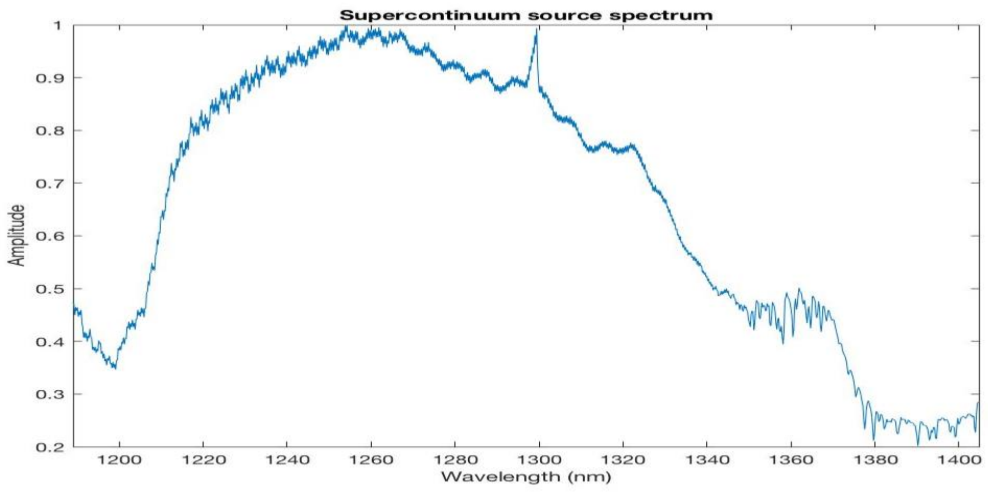


Figure 7. Supercontinuum laser spectrum with a 135 nm bandwidth.

2.2 Michelson Interferometer

Interferometers work by splitting an optic signal, so its components propagate through two different paths, the so-called reference and sample arms, then reconstructing the returning optic signals to create an interference signal. The interference signal can be used to obtain information about the laser source and the different path lengths that it has travelled through. An amplitude-line, or A-line, is an interference signal obtained in time or frequency domain that contains information about the sample in the form of an intensity profile describing the structures within the sample. These are the building-blocks of OCT imaging and are stacked together to form 2-dimensional images or 3-dimensional volumes of the sample. A broadband laser source is used in OCT imaging because an interference signal can only be obtained if the reference and sample arm path lengths are equal to within the coherence length of the source. Then, the smallest structure that can be resolved in the axial direction is equal to the coherence length of the source.

The fiber-based components of the Michelson interferometer, Fig 8, are a wideband 2x1 beamsplitter (TW1300R5A1, Thorlabs) with 50:50 split ratio and an optical circulator (CIR-1310-50-APC, Thorlabs) that allows the interference optical signal to travel to the spectrometer rather than return back to the source. These components use single-mode (SM) fiber. In order to obtain an interference pattern, the lengths of the sample and reference arms must be matched. Although the individual fiber lengths of the beamsplitter are matched, there is an additional path on the sample arm, through the collimator and focus lens, that must be matched at the reference arm. The additional path needed is achieved with a second collimator and a reflective gold mirror on a variable-length optic cage. Furthermore, the optical power of the reference and sample arm must be balanced so that one signal does not envelope the other. A good quality interference pattern is achieved if the intensities of the sample and reference arms are in the same order of magnitude.

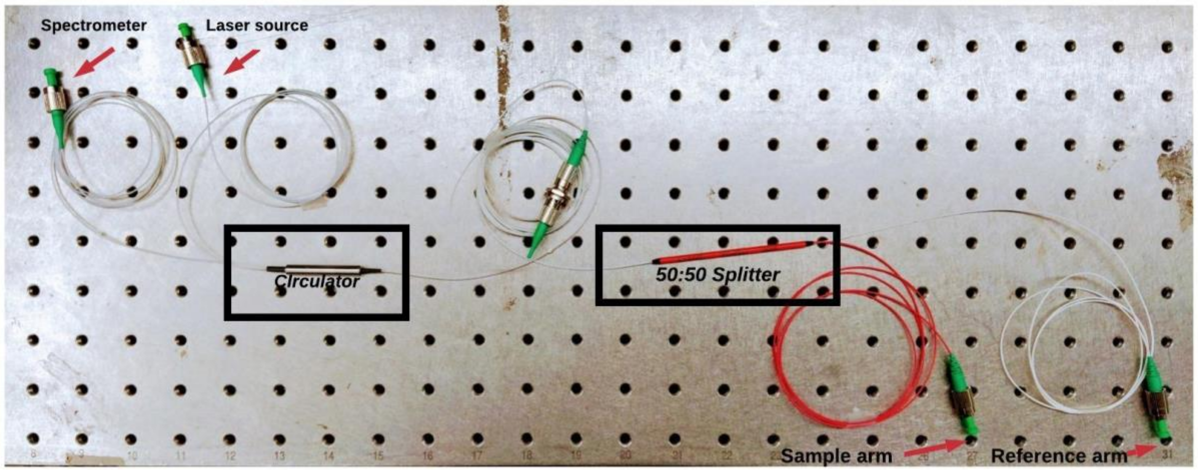


Figure 8. Fiber-based circulator and beamsplitter components of the Michelson interferometer.

2.2.1 Single-Mode (SM) Optic Fiber

Optic fibers designed to allow propagation of only the fundamental mode (LP_{01}) are said to be single-mode (SM) fibers. In contrast with multi-mode fiber, single-mode fibers have a smaller core diameter. A common core diameter value used in the industry for SM fibers is 8.2 μm. SM fibers are designed with a small core radius and small refractive index between core and cladding. These fibers do not suffer from intermodal dispersion and have lower propagation losses than multi-modal fiber. They find use in communication and medical imaging applications.

2.2.2 Beamsplitter (TW1300R5A1)

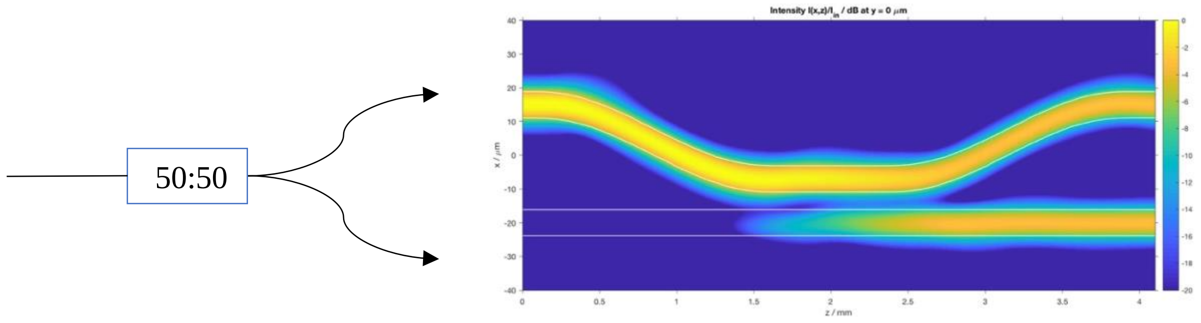


Figure 9. Graphical depiction and simulation of a beamsplitter or coupler with a 50:50 power split ratio. Simulation performed in MATLAB® using the BeamLab toolbox.

Fiber-based beamsplitters are achieved through the Fused Biconical Taper (FBT) process. This industrial process fuses two fibers together so that their cores are very close to each other along a given length. The close proximity of the cores allows the evanescent wave from either core to leak into the other, Fig 9. The length and proximity of the fuse controls the ratio of the energy shared by the cores. The TW1300R5A1 optic coupler from Thorlabs Inc. has a 50:50 split ratio and is used to separate and recombine the laser beam. This single-mode tap has a center wavelength of 1300 nm and bandwidth of 200 nm.

2.2.3 Circulator (CIR-1310-50-APC)

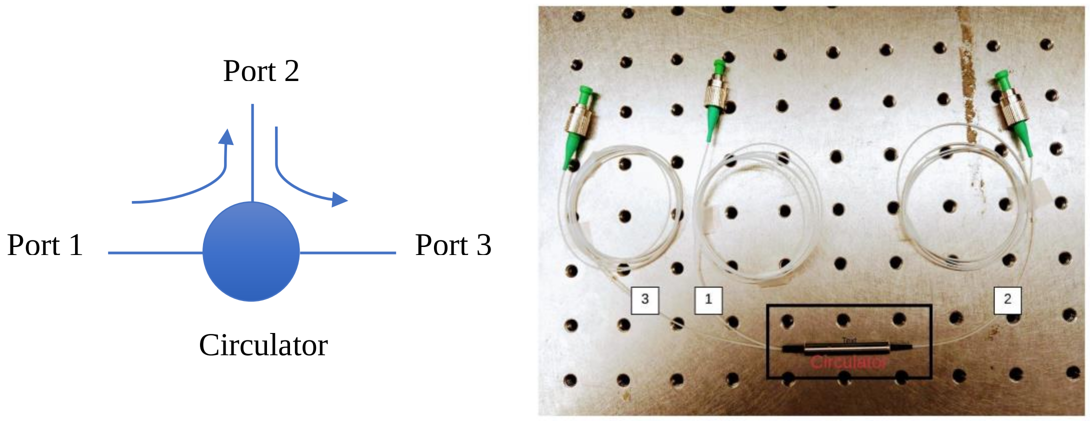


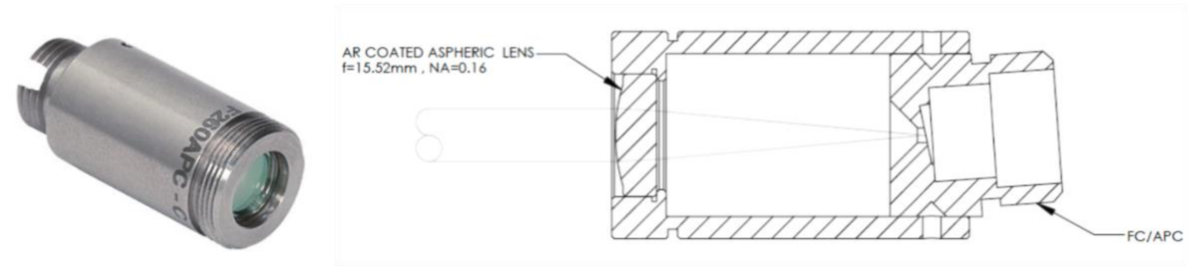
Figure 10. Optical circulator diagram and circulator component (CIR-1310-50-APC) from Thorlabs.

Optic circulators are three-port devices that control the direction light travels. They are fabricated in a manner similar to fused fiber couplers. However, when light enter port 1 of a circulator the intended purpose is to minimize the power being transferred to port 3 while maximizing the power coupled to port 2. Accordingly, when light enters port 2 the circulator couples the light to port 3. The CIR-1310-50-APC circulator from Thorlabs was used to redirect the interference beam into the spectrometer, Fig 10. The circulator has a damage threshold of 500mW with a maximum insertion loss of 1.6dB and operating wavelength range of 1280-1400 nm. The circulator is the bottleneck for axial resolution with a bandwidth of 120 nm. This bandwidth is smaller than the source, beamsplitters, and spectrometer’s bandwidth, making the circulator the limiting factor for axial resolution.

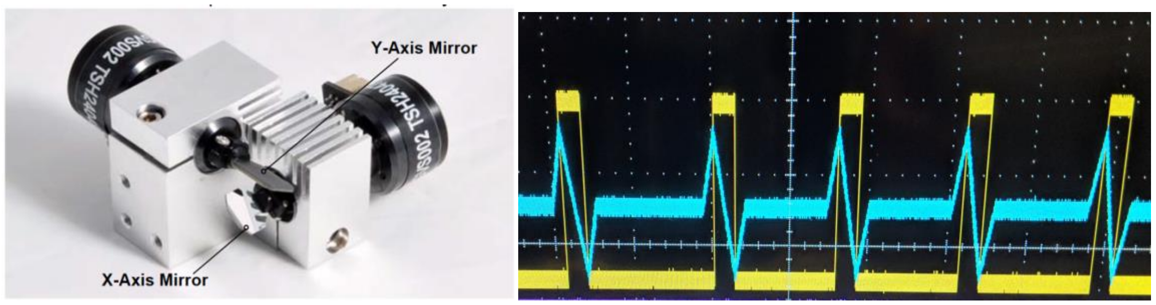
2.2.3 Solid-state optic components (collimator, reflective gold mirror & galvo-controller mirror, focus lens)

The fiber-based components need to be complemented with solid-state optic components to complete the interferometer. After the laser beam splits, its components exit the fiber and are reflected by a reference mirror and backscattered by the sample. Beam collimators are optic devices commonly used as the output of a single mode fiber, necessary to maintain the beam radius constant while propagating through free-space. The F260APC collimator uses an aspherical lens, as seen in Fig 11a, so that the curved wavefronts become flattened. At the reference arm, the collimator is followed by a gold mirror to reflect the beam back into the collimator. These components are mounted on a variable-length optic cage. At the sample arm, the collimator is followed by a galvo mirror, Fig 11b, and a plano-convex lens (LA5817), Fig 1ac, with a diameter of 2.5cm and a focus length of 100mm, used to sweep and focus the collimated beam, respectively. By sweeping the beam across the sample, multiple A-lines are acquired to create an image, or B-scan. Furthermore, the scanning of the galvo mirror needs to be synchronized with the acquisition camera to avoid split frames. The focus lens concentrates the beam on the sample so that sufficient optical scattering is created to be picked up by the camera.

[a]



[b]



[c]

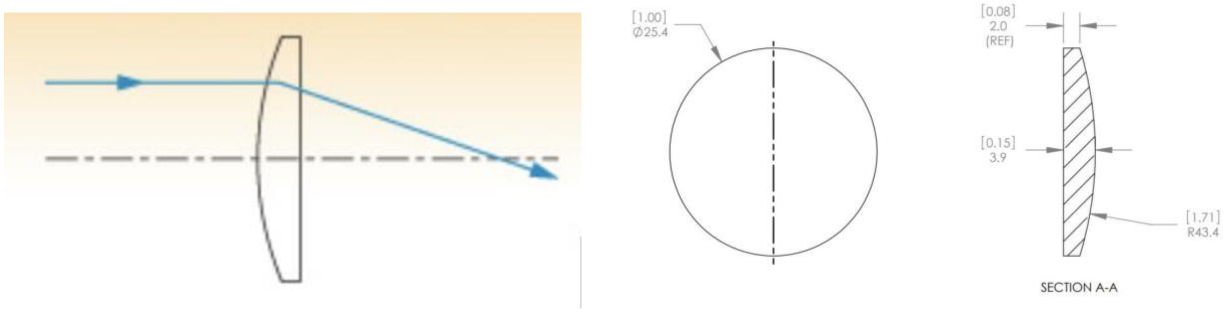


Figure 11. Solid-state optic components. [a] Collimator F260APC with a focal length of 15.36 mm and numerical aperture of 0.16. [b] Dual-axis galvo mirror with driving signal and camera sync signal. [c] Plano-convex lens to focus the beam on the sample.

2.3 Spectrometer

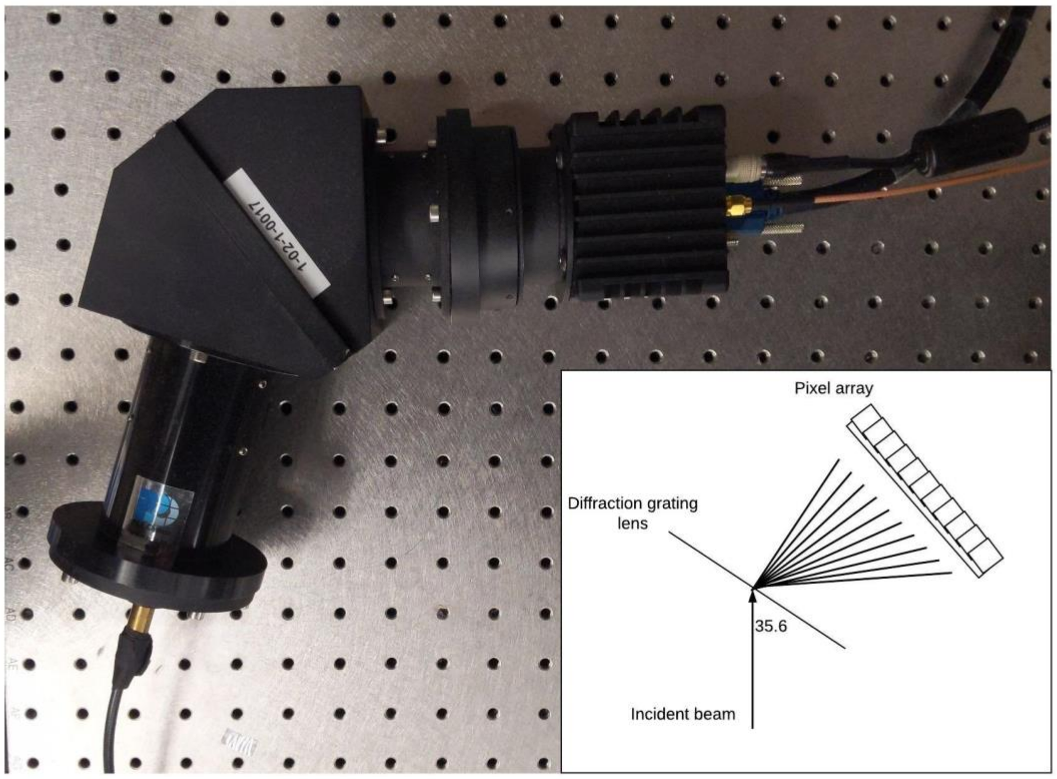


Figure 12. Custom-built spectrometer from P&P Optica.

The spectrometer was custom built by P&P Optica for OCT imaging at 1300nm. A transmissive diffraction grating element and a high-speed camera, Fig 12, are used to sample the interference signal in k-space. A diffraction grating lens with 892 lines/mm grating frequency placed at 35.6 degrees angle relative to the incident beam is aligned with the camera array sensor. The optical beam incident on the diffraction grating element is separated into its wavelength components and each of these beams are detected by one of 1024 sensors. Diffraction grating elements are fabricated by periodically making ridges on a lens (transmissive grating) or mirror (reflective grating). The period of the ridges or grating defines the angle at which an incident beam will be diffracted into multiple beams.

$$m\mu\lambda = \sin\theta_i - \sin\theta_d$$

- m is the grating diffraction order
- μ is the grating frequency in lines/mm
- λ is the incident beam wavelength
- θ_i is the incident beam angle
- θ_d is the diffracted beam angle

Equation [8]. Diffraction grating equation.

The grating's diffraction angle is a nonlinear function of the propagation constant, k, hence the signal recorded by the camera is unevenly spaced in k-space, Eq [9]. This is important to note because in order to reconstruct the image it is necessary to perform a Fourier transform operation, for which a linearly spaced k-domain is needed for best results. Therefore, a resampling and interpolation of the interference signal is necessary before the performing the Fourier transform operation.

$$\lambda_i = 1188.89 + 0.2268 * x - 8.2 * 10^{-6} * x^2 - 6.99 * 10^{-9} * x^3 \qquad x = 1,2,3,\ldots,1024$$

Equation [9]. Nonlinearly spaced wavelength values incident on each pixel of the sensor array as per the manufacturer's calibration data.

The LDH2 from Unlimited Sensors is a high-speed camera designed specifically for OCT applications. It has a 1024-pixel InGaAs sensor array and a maximum readout of 91kHz. The LDH2 camera is connected to a PCIe-1427 frame grabber through a MiniDeltaRibbon 26-pin Camera Link cable. The Camera Link cable is a 26-pin bus that uses a low-voltage differential signaling (LVDS) for high-speed communication. The frame grabber (PCIe-1427) is an image acquisition device manufactured by National Instruments that uses the Peripheral Communication Interface Express (PCIe) for high-speed communication.

2.4 Data Acquisition and Processing

The Laboratory Virtual Instrument Engineering Workbench, LabVIEW, from National Instruments is used to acquire and process the data sent by the camera to the frame grabber card. The NI-IMAQdx library is used to handle camera related tasks, establishing a connection and managing memory. The NI-DAQmx library is used to generate waveforms and trigger signals. The raw data acquired by the frame grabber is processed to obtain the intensity profile and compute the peak intensity, following the block diagram shown in Fig 13.

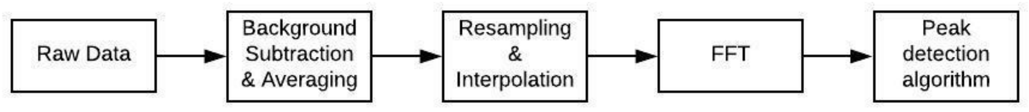

Raw Data
Background Subtraction & Averaging
Resampling & Interpolation
FFT
Peak detection algorithm

Figure 13. OCT data processing block diagram

2.4.1 Raw data, background subtraction, and averaging.

The frame grabber obtains 1000 A-lines at a rate of 91 fps resulting in a 1000x1024 (row-by-column) matrix of integers, $I_{i,j}(k)$, where each row represents an optic intensity signal in k-space as defined by the spectrometer. The average of the matrix is the background noise of the signal. The background noise modulates the information carrying signal and needs to be subtracted. This background noise corresponds to the reflectivity of the reference mirror. It can be computed by averaging each column along the matrix to obtain a single 1x1024 array that is subtracted from the raw data matrix. After background subtraction, Fig 14, the matrix is divided into ten parts and each submatrix averaged resulting in a 10x1024 matrix. This step is taken to reduce the data set for the next processing steps and has the additional benefit of ensuring optic signal stability.

$$I_{mean} = \frac{\sum_{i=1}^{1000} I_{i,j}}{1000} \qquad \text{for } j = 1, 2, 3, \ldots, 1024$$

Equation [10]. Mean signal calculation.

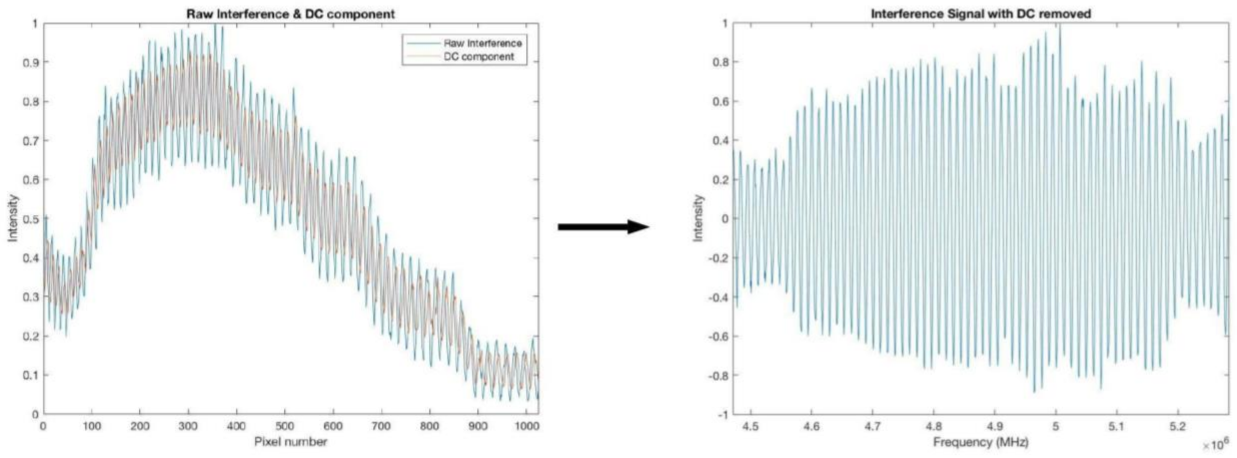


Figure 14. Raw data with background noise component. [b] Interference signal after background noise subtraction.

2.4.2 Resampling and Interpolation

In OCT systems, the grating separates the incident beam into different wavelengths. Each pixel in the sensor array measures the beam at a given wavelength, as a result the analog-to-digital conversion is done in the wavelength domain of the signal. However, the FFT operation expects the signal to be evenly spaced in the frequency domain or k-space. If one knows the spectral distribution incident on the sensor array, given by Eq [9], it is possible to resample the signal to linearly spaced k values, using Eq [10]. Interpolation calculation gives the new optic intensities associated with the newly resample k values. Cubic or spline interpolation fits a third-degree polynomial between the two interpolation points. While resampling and interpolation methods considerably decrease signal processing speed, it is a critical step to improve image resolution, Fig 15.

$$k_i = \frac{2\pi}{\lambda_i} = 2\pi\left(\frac{1}{\lambda_{max}} + \frac{i}{1024-1}\left(\frac{1}{\lambda_{min}} - \frac{1}{\lambda_{max}}\right)\right) \quad i = 1,2,3,\ldots,1024-1$$

Equation [11]. Linearly spaced k values used to interpolate the new intensity values.

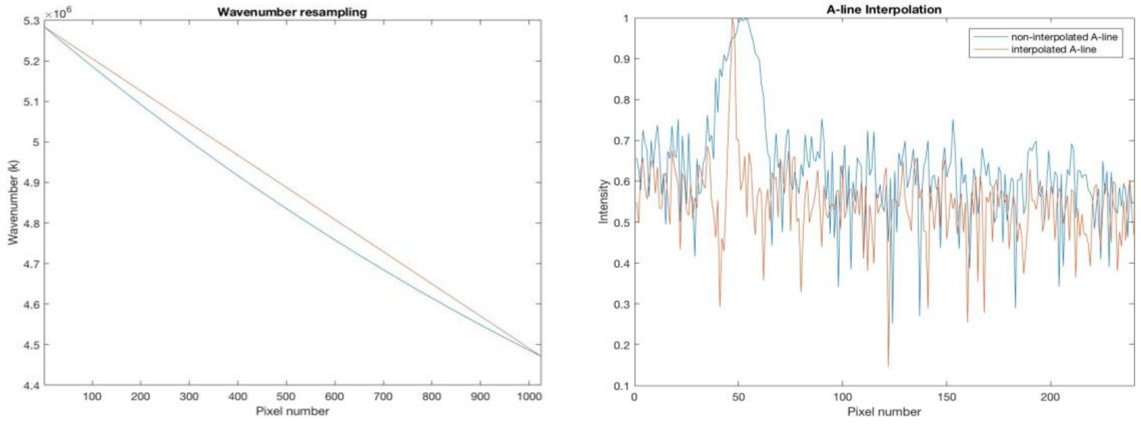


Figure 15. Linearly resampled domain and interpolated A-line.

2.4.3 Fourier Transform

The Fourier transform is used to convert the interference signal from frequency to time domain. The diffraction grating lens, previously described, performs a Fourier transform on the beam by decomposing it into its component wavelengths. Hence, it is necessary to reverse the operation. The magnitude of the Fourier transform output contains the intensity depth profile of the sample.

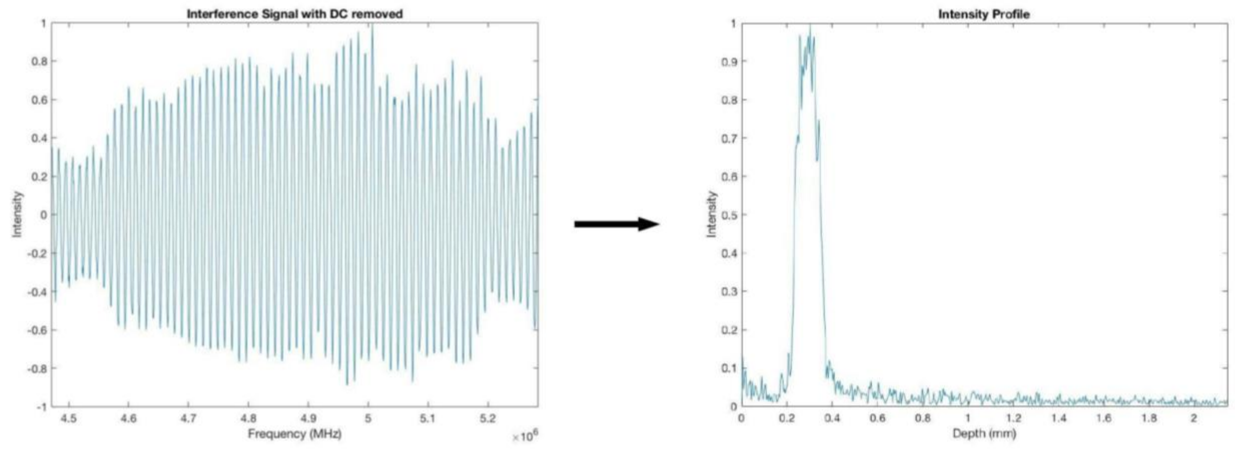


Figure 16. Fourier transform on the interference signal extracts the sample structural information.

2.4.4 Peak detection algorithm

Following the Fourier transform operation, the position along the depth axis of the peak intensity is computed. In general, the highest intensity on an A-line is at the surface of the sample. Therefore, if the surface of a sample is desired to be located, a peak detection algorithm suffices. In practice, the instability of the optic signal is a problem that causes the peak to disappear. Taking the average of a number of A-lines, as previously described, and performing the peak detection ensures that there is no data loss.

2.5 OCT end-effector

The construction of the interferometer as the robot end-effector went through two design stages to ensure that the requirements were met, and revisions based on surgeon-feedback were made. The first prototype was built with off-the-shelf optic posts and rods on an optic breadboard. This heavy setup accommodates the galvo cage and a trio of bulky aiming lasers, Fig 17. In this first prototype, the variable length optic cage at the reference arm was not placed on the end-effector and neither was the fiber-based circulator and beamsplitter. The second prototype of the OCT end-effector is presented in Fig 18. It was design in AutoCAD and 3D printed using a MakerBot

Replicator 2X with polylactic acid (PLA), a common thermoplastic used in additive manufacturing. The end-effector was constructed to contain the the interferometer and aiming lasers. The reference arm consisting of a collimator, gold mirror, and variable length optic cage is mounted at the top of the end-effector. The sample arm with the collimator, galvo-controlled mirror, and focus lens is located at the bottom of the end-effector pointing downwards to the sample. The aiming lasers are three 650nm red lasers oriented so that their beams converge on the focus spot of the sample arm's focus lens. It is used to give the surgeon a broad sense of where the imaging workspace is. Needed because the OCT laser is not in the visible spectrum and the imaging workspace is too small for easy manual targeting. Therefore, an aiming laser system in order to help orient the surgeon while guiding the robot towards the imaging workspace is included. The interferometer, in its entirety, is accommodated on the end-effector in order to avoid compromising its calibration and avoid disturbances in the fiber. The enclosing of the fiber-based optical components (i.e. circulator and splitter), located in the posterior of the tool, was designed sufficiently large to minimize bend losses through the fiber. The laser source and spectrometer were not mounted on the robot for practical reasons (i.e. size, weight). The geometry and profile of the end-effector was considered during design so as to not interfere with robot motion. The physical dimensions were measured, and its weight taken and imported in the robot software. These values were used to calculate the end-effector center of mass in a built-in robot application that uses its joint torque sensors to compute the values. Safety-related integration of the end-effector requires a calibration process to ensure that all physical parameters (i.e. dimensions, weight, center of mass) are known to the robot controller, without this information, the robot cannot be used in a safe collaborative manner.

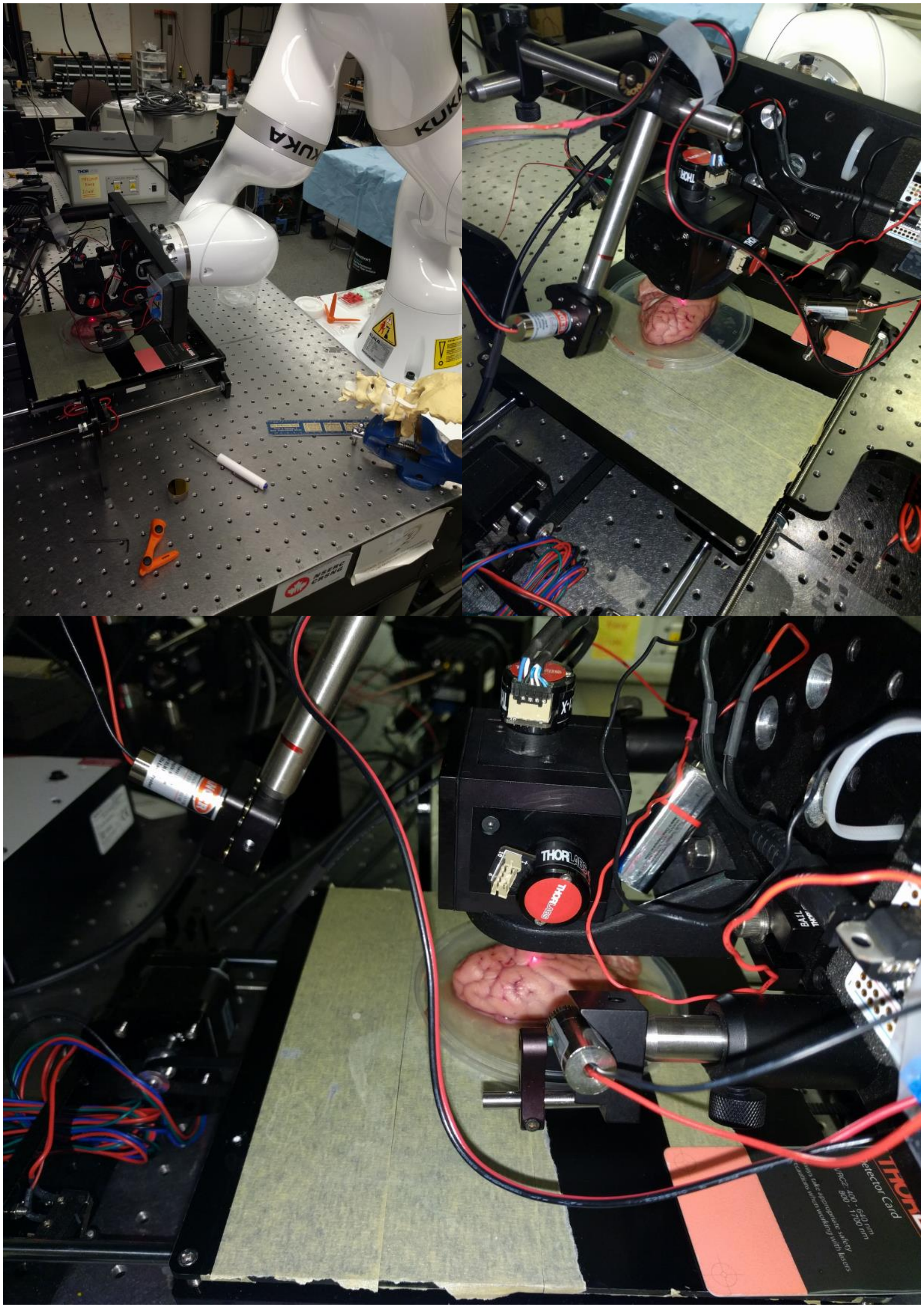
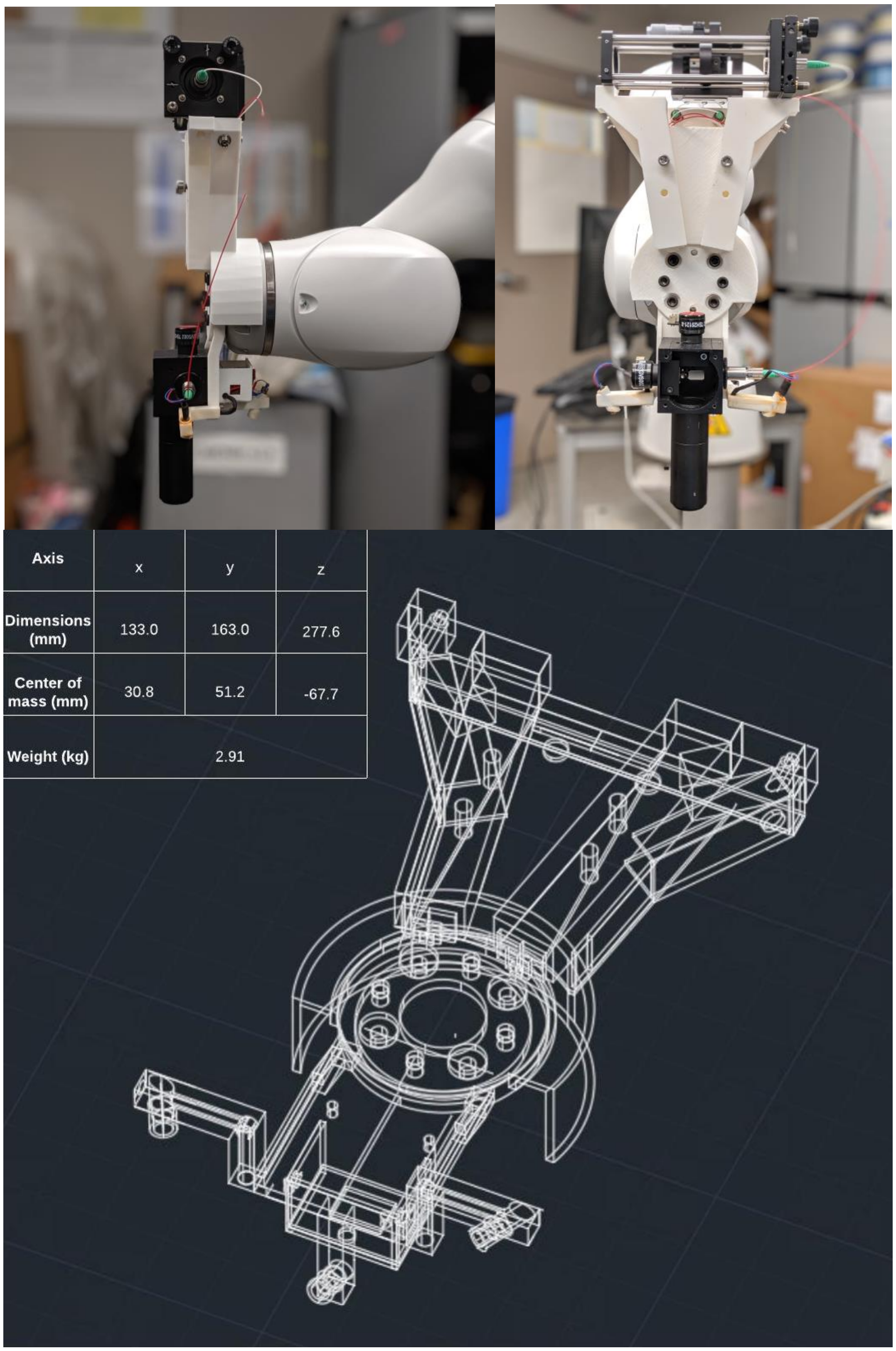


Figure 17. First-generation prototype of the OCT end-effector.



Axis	x	y	z
Dimensions (mm)	133.0	163.0	277.6
Center of mass (mm)	30.8	51.2	-67.7
Weight (kg)	2.91		

Figure 18. Robot end-effector for OCT imaging with interferometer and aiming lasers.

2.6 Imaging Performance

A Gaussian beam refers to the function of the optical intensity across the transverse plane of beam propagation to be Gaussian. Gaussian beams are the lowest-order self-consistent field distribution in an optical resonator, for that reason the majority of lasers generate Gaussian beams. Mathematical calculations for Gaussian beam propagation are simpler since the intensity profile at any point on the beam is Gaussian with only the radius of the beam changing. In addition, the intensity profile of single-mode fibers is also Gaussian-like so Gaussian beams can be launched into the fiber with high-efficiency and propagation calculations are straightforward. The Gaussian intensity profile of the collimated beam before entering the focus lens at the sample arm of the interferometer is shown in Fig 19. The beam power was detected with an optical power sensor (S144A from Thorlabs) to be 82mW (P = 82mW) and the diameter was measured with a beam profiler (BP109-IR from Thorlabs) to be 2.52 mm ($d = 2.52mm$).

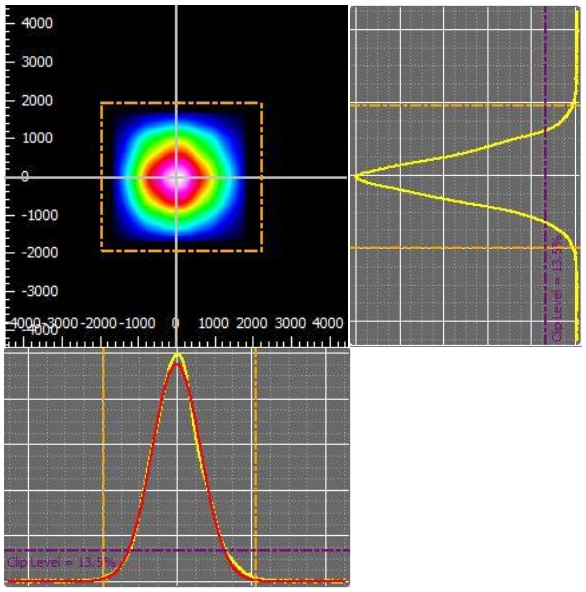


Figure 19. Single-mode Gaussian beam profile.

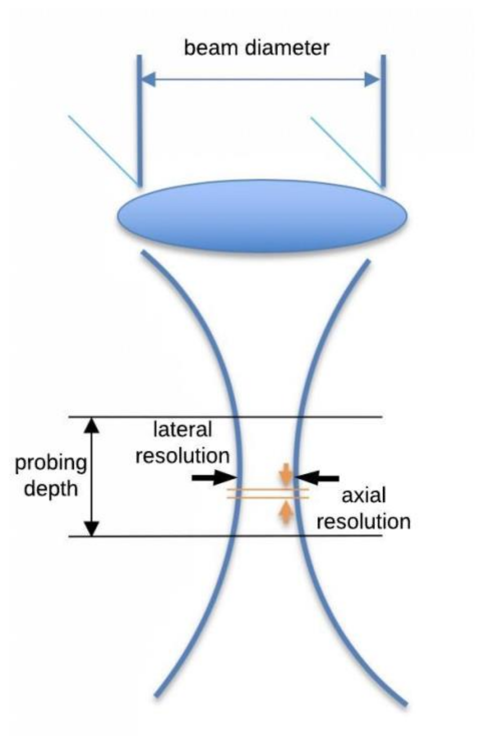


Figure 20. OCT imaging characteristics of the probing beam: probing depth, lateral resolution, and axial resolution.

Once the beam is transmitted through the focusing length it is concentrated into a focus spot, shown in Fig 20. The diameter of the focus spot was also measured with the beam profiler to be $66\mu m$ $(2w_0 = 66\mu m)$. The peak intensity at the focusing spot can be calculated through:

$$I = \frac{2P}{\pi {w_0}^2} = 47.9\, {}^{MW}\!/_{m^2}$$

Equation [12]. Peak optical intensity at the focusing spot of the Gaussian beam

The Rayleigh range of a laser beam is the distance from the focus spot at which the beam radius becomes $w_0\sqrt{2}$. It is also called the depth of focus because the optical intensity is greater, and the rays of light converge at this point to provide a more focused, less blurred image. The value can be calculated from:

$$z_R = \frac{\pi {w_0}^2}{\lambda_0} = 2.6mm$$

Equation [13]. Rayleigh range of the Gaussian beam.

However, this value does not provide the whole picture in terms of probing depth for frequency-domain OCT imaging. The probing beam is focused at the sample and the coherence matching point is located at the focal point of the sample lens. The actual probing depth for a frequency-domain OCT imaging system depends on the wavelength resolution of the spectrometer. Eq [13] is a relationship for the expected probing depth as a function of laser spectrum width, $\Delta\lambda$, and the number of elements in the sensor array, N.

$$z_{max} = \frac{1}{4}\frac{{\lambda_0}^2}{\Delta\lambda}N \approx 2.1\ mm$$

Equation [14]. Probing depth.

Lateral or transverse resolution is a function of the optic properties of the focus lens at the sample arm of the OCT system. The focus spot size, previously measured with the beam profiler, gives lateral resolution. As shown in Eq [5], a lens with a high numerical aperture ($NA = {}^{d}/_{f}$) creates a small spot size, improving lateral resolution.

$$\Delta x = \frac{4\lambda_0}{\pi}\frac{f}{d} = 66\ \mu m$$

- f is the focus length of the lens

- d is the diameter of the collimated beam

Equation [15]. Lateral resolution of the imaging system defined by its spot size.

Image quality depends on the axial and lateral resolution of the optic system. Resolution is dependent on the laser source and optic configuration, Fig 20. For a laser source with a Gaussian spectral distribution the temporal coherence length defines the axial resolution. The laser source spectrum is correlated to the axial resolution through $\lambda^2/\Delta\lambda$. The coherence length and hence the axial resolution can be calculated through:

$$l_c = \Delta z = \frac{2ln(2)}{\pi}\frac{{\lambda_0}^2}{\Delta\lambda} \approx 5.6\mu m$$

Equation [16]. Coherence length of the laser source and axial resolution of the imaging system.

.

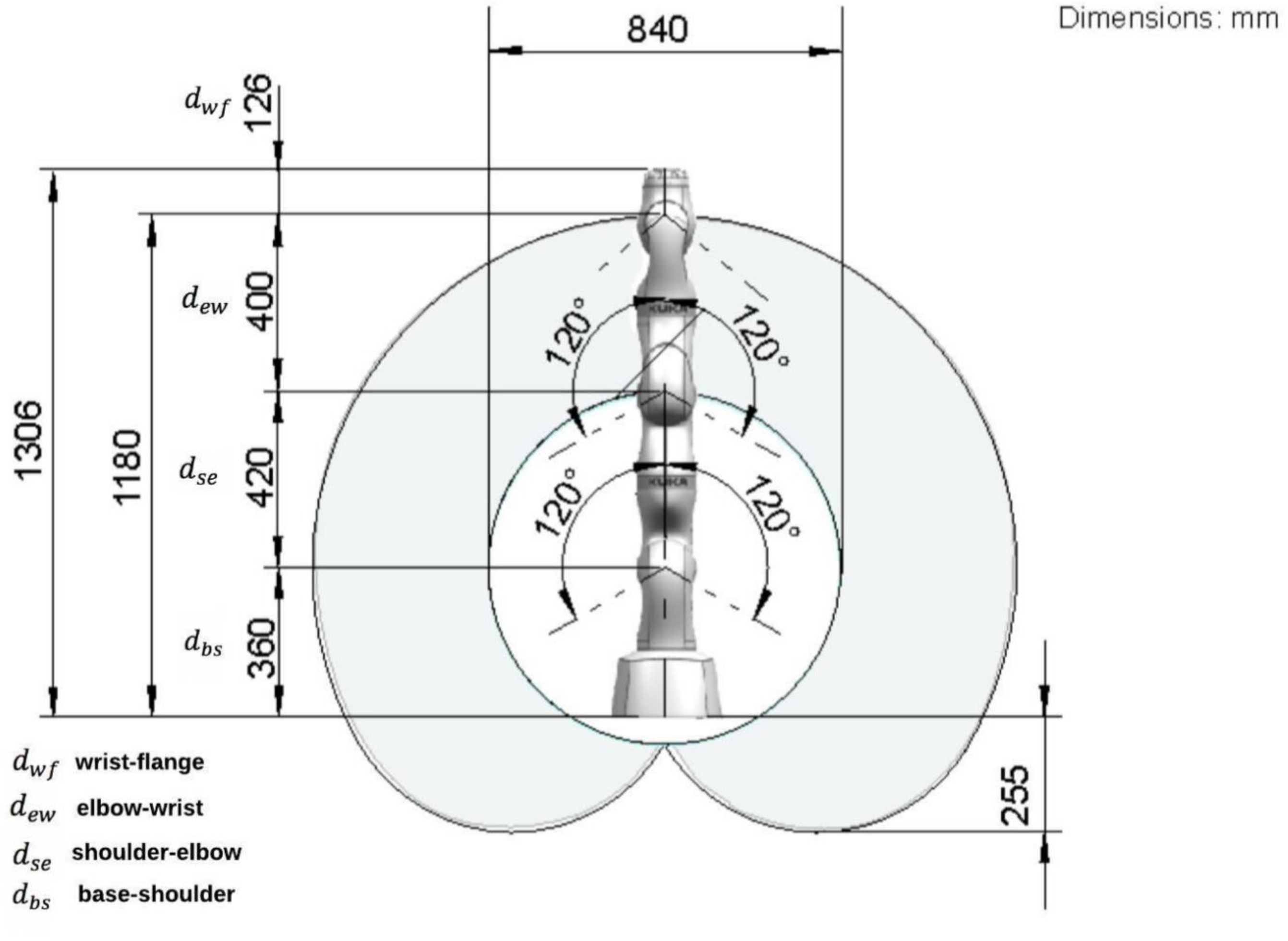
3. System integration, Experiment methodology, and Results

3.1 LBR Med – Collaborative robot & safety-oriented programming

The complexity of the field of robotics crosses traditional engineering boundaries and puts mechanical, electrical, and computer engineering together. The technology behind computer controlled mechanical arms was born in the 1950s with the design, creation and successful commercialization of the Unimate robotic manipulator. The invention of the robotic manipulator by George Devol and its deployment in General Motors factories by Joseph Engelberger started the industrial robot age in automation. The first-generation position-controlled robotic arms were extremely useful performing hazardous repetitive tasks within a fixed industrial environment. Today, the use of industrial robots has helped optimize factory automation processes to the point where the adoption of new robot technology is needed to continue the growth and discover novel solutions to existing problems. Out of this need and to this end, torque control and navigation technology is emerging to create the concept of collaborative robotics with the goal of deploying robots in unstructured, dynamic environments to work in the presence of humans. Constraints previously placed on robots with regards to human interaction are being removed thanks to improved sensors as well as better control, tracking, and navigation software being added to this new generation of collaborative robots. The collaborative robot age is moving the robots from industrial factories to hospital, airports, and hotels.

KUKA AG (Augsburg, Germany) became a pioneer in the collaborative robot space after a decade-long research and development collaboration with the German Aerospace Institute that resulted in the LBR I in 1995 and LBR II in 2000. These 7 degrees-of-freedom (DOF) robotic arms were meant to explore the light weight robotics with high payload-to-weight ratio. Ultimately, the rights to the robot technology were licensed to KUKA after the development of the LBR III in 2004. KUKA kept iterating their light-weight collaborative robot technology and in 2013 released the LBR iiwa (Light Weight Robot - intelligent industrial work assistant) to explore human-robot collaboration. Consequently, KUKA recognized a growing trend in medical applications and released the LBR Med, the world's first certified robot to be integrated into medical devices, in 2017. The LBR Med builds on the earlier iiwa version, with additional medical devices regulatory compliance. The International Electrotechnical Commission (IEC) is the recognized organization that prepares standards for a number of technologies, including medical devices hardware and software. *IEC60601 Medical Electrical Equipment* and *IEC62304 Medical Device Software* outline requirements for medical equipment that were observed. Compliance with IEC60601 requires a routine brake test to ensure that each physical brake can hold the maximum allowed torque on its joint. The programming paradigm of the LBR Med meets the requirements of the *IEC62304 Medical Device Software* document by implementing separate motion and safety controllers. For example, commanding a robot to a position requires a motion command to the desired position while using a safety controller command to guarantee the path to be followed towards said position. In other words, the robot application program is implemented using the motion controller while engaging the safety controller for redundancy. The safety controller monitors the system in terms of position, velocity, and force. In the event of a fault, meaning that one of the safety parameters is exceeded, the single fault safety required by the regulatory bodies can only be guaranteed for the behavior of the safety controller. The operating system running on the robot was developed by the manufacturer and named Sunrise. The Sunrise OS allows the control and programming of the robot to be separated. Position, velocity, force, and impedance control, as well as gravity compensation and null space positioning tasks can be programmed on the robot at a high-level through the development environment, Sunrise.Workbench, using Java.

Robot arms are defined by their number of joints, the different types of joints, and length of the links between joints. Fig depicts the geometric structure of the robot arm and its S-R-S-R-S-R-S (spherical-rotational-spherical) kinematic structure. The forward kinematics matrix of a robot combines the geometry and kinematics to obtain a relationship between robot joint angles and Cartesian space. We can use the Denavit-Hartenberg (DH) convention to assign a coordinate frame to each robot joint using four parameters to relate the current joint, j_{i-1}, to the next, $\mathrm{j_i}$. The DH convention describes an articulated sequence of joints by defining the pose of a link frame with respect to the previous link frame, starting from the most distal joint, $\mathrm{j_7}$. The transformation between frames is described by four parameters that correspond to a series of rotations and translations. The first transformation is a rotation about the z-axis by angle theta, $\theta_i,$ followed by a translation along the z-axis corresponding to the length of the link joining the joints, d_i, and a second translation along the x-axis. Lastly, a second rotation along the x-axis by angle alpha, α_i, corresponds to the orientation of the new frame z-axis with respect to the previous frame z-axis. The DH parameters for the LBR Med, shown in Table, can be used to create the transformation matrix that relates the reference frame of each joint to the next. The product of these matrices results in the forward kinematic matrix defining the pose of the robot arm on Cartesian space as a function of joint angles, θ_i. The 7 DOF configuration gives a kinematic redundancy so that the pose can be maintained while moving the end-effector on Cartesian space, analogous to how the human arm allows elbow motion while holding wrist pose. Each joint has a strain gauge-based torque sensor and dual position sensors, with measurements made at a rate of 3kHz. Redundant position sensors increase both accuracy and safety of the robot. Robot kinematics are computed in the cabinet controller at a rate of 1kHz.



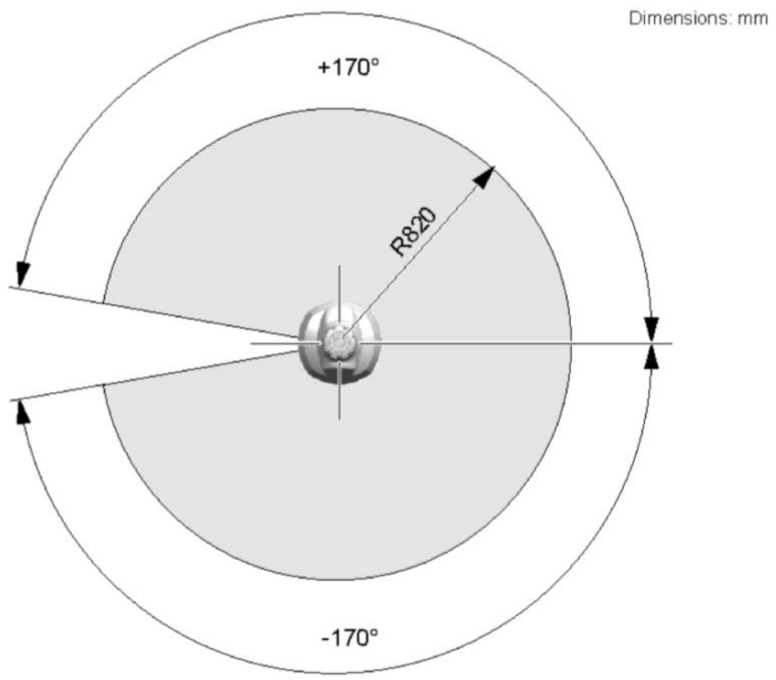


Fig 21. LBR Med 14 R820 configuration space information and workspace envelope.

$$R_z(\theta_j) = \begin{matrix} \cos\theta_j & -\sin\theta_j & 0 \\ \sin\theta_j & \cos\theta_j & 0 \\ 0 & 0 & 1 \end{matrix} \qquad R_x(\alpha_i) = \begin{matrix} 1 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i \\ 0 & \sin\alpha_i & \cos\alpha_i \end{matrix}$$

$$T_z(d_i) = \begin{matrix} 0 \\ 0 \\ d_i \end{matrix} \qquad T_x(a_i) = \begin{matrix} a_i \\ \mathbf{0} \\ \mathbf{0} \end{matrix}$$

Joint ($\boldsymbol{j_i}$)	$R_x\ (\boldsymbol{\alpha_i})$	$T_x\ (\boldsymbol{a_i})$	$T_z(\boldsymbol{d_i})$	$R_z\ (\boldsymbol{\theta_i})$
$\boldsymbol{j_1}$	-π/2	0	d_{bs}	θ_1
$\boldsymbol{j_2}$	π/2	0	0	θ_2
$\boldsymbol{j_3}$	π/2	0	d_{se}	θ_3
$\boldsymbol{j_4}$	-π/2	0	0	θ_4
$\boldsymbol{j_5}$	-π/2	0	d_{ew}	θ_5
$\boldsymbol{j_6}$	π/2	0	0	θ_6
$\boldsymbol{j_7}$	0	0	d_{wf}	θ_7

$$^{j-1}T_j = R_z(\theta_j)T_z(d_j)T_x(a_j)R_x(\alpha_j)$$

$$^{j-1}T_j = \begin{matrix} cos\theta_j & -sin\theta_j cos\alpha_i & sin\theta_j cos\alpha_i & a_i cos\theta_j \\ sin\theta_j & cos\theta_j sin\alpha_i & -cos\theta_j sin\alpha_i & -a_i sin\theta_j \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$^0T_7 = {}^0T_1\,{}^1T_2\,{}^2T_3\,{}^3T_4\,{}^4T_5\,{}^5T_6\,{}^6T_7$$

Equation [17]. Elementary transformation matrices and Denavit-Hartenberg (DH) parameters to obtain the forward kinematics matrix. Image modified from the robot manual.

Human-robot collaboration is becoming the basis of novel surgical techniques. However, regulatory standards in surgical robotics is often a concern when issues of liability are raised. Legal, regulatory, and ethic arguments have to be explored in order to understand the technology and its potential adoption by the medical community [25]. To this end, government bodies, education institutions, and private companies are collaborating to lay out a direction for future research and development of collaborative robots for medical applications. The European Union has funded the SAFROS project with the goal of adapting an existing safety framework to the requirements of robot-assisted surgery [26]. RoboLaw is another project aiming to build a legal framework around which they can regulate robotics. The project ran from 2012 to 2014 led by a group of researchers from Italy and was funded by the 7th Framework Program (FP7), the European Union’s research and development grant program. It resulted in a report outlining legal and ethical concerns pertaining novel technology in the field of robotics [27] [28].

The adoption of collaboration robots is a slow process full of technology setbacks and regulatory hurdles. In order to remove the literal fence around the robot, years of risk assessments and development of safety frameworks for collaborative robotic tasks have been performed. The International Standards Organization (ISO) is working on providing guidance and outlining requirements for safe collaborative robot tasks. Operator safety frameworks marks the difference between collaborative robots and non-collaborative robot. *ISO 10218-1&2 Robots and robotic devices* is the standard that specifies common risks associated with robots and outlines risk elimination methods. *ISO/TS 15066 Collaborative Robots* builds on the aforementioned standards but adds information and requirements specific to human-robot collaboration tasks. The key idea behind ISO/TS15066 is that “if contact between human and robots is allowed … then that contact shall not result in pain or injury”. This standard defines four types of collaborative operations (*Safety-monitored stop, Hand-guide mode, Speed and separation monitoring, Power and force* *limiting)* and gives information and guidelines to maximize safety during such operations. It provides comprehensive guidance for those conducting risk assessment of collaborative robot applications, and outlines speed, force, and pressure permissible during such applications. The need to determine force and pressure threshold values has risen as more robots are used in collaborative tasks. However, there is a practical difficulty and ethical concerns that need to be addressed in order to obtain those numbers. A number of studies have been approved and carried out to determine pain threshold values using both phantoms and human volunteers under dynamic, quasi-static, and static loads. Nevertheless, challenges for safety-critical robots remain. The author designed a safe collaborative workspace within acceptable parameters according to the best present knowledge of the state of the art [29] [30].

Programming of the LBR Med robot is done through the Sunrise.Workbench development environment, through which, robot motion, safety, and tool data can be loaded on the controller for processing and monitoring. In order to introduce a collaborative robot into a surgical workflow, the application/software running on the robot must be safe and flexible. This gives the surgeon control over the process so different problems can be solved while guaranteeing robot operation safety throughout. Robot application workflow design requires a careful examination of the risk factors to the surgeon, patient, and surgical staff, as well as any other equipment present in the operating room. We outline a collaborative workflow with an *adaptive safety monitoring* protocol placed to reduce risk. As the level of interaction with the surgeon and the proximity to the patient increases, the safety parameters become more constrained. Cartesian velocity and collision detection parameters are modified to the level of risk inherent in the current task. In order to meet requirements outlined by *IEC62304*, the collaborative robot software has separate motion and safety controllers, where the safety controller is a redundant software to monitor the motion controller. As shown in Fig 22, at each stage of the robot workflow there is a motion command

and corresponding safety tasks are enabled. In the case where the robot needs to be held in a pose while the surgical staff enables the imaging system, the motion controller does not guarantee the holding of the position and the safety controller is used as a redundancy to address any faults that may occur. The safety controller parameters (i.e. position, velocity, and force) are redundant to the application running on the motion controller.

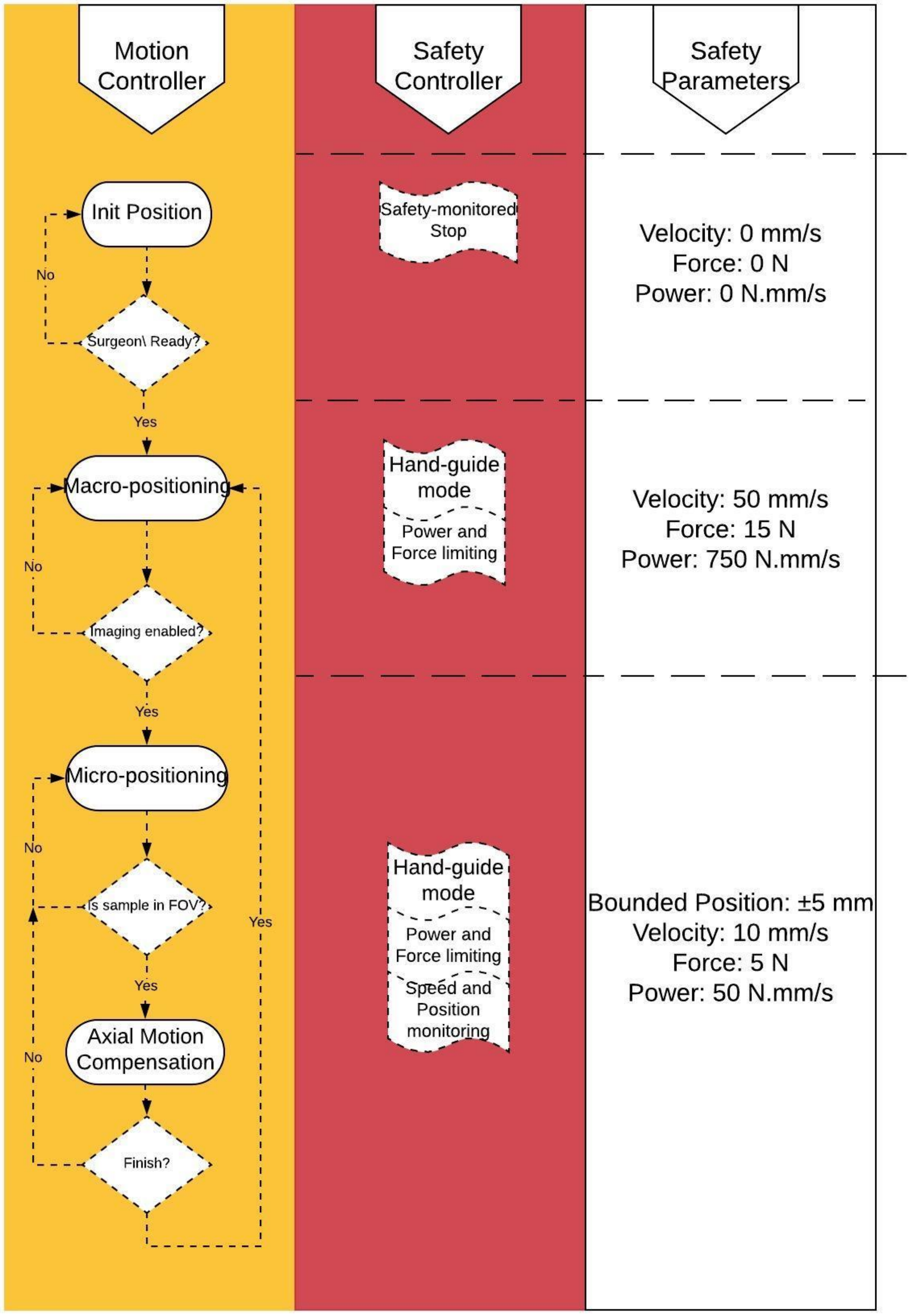


Figure 22. Safety-oriented workflow design for intraoperative OCT imaging.

The operation of a robot arm during surgery needs to be done by an experienced surgeon who understands the capabilities of the system. Robot workspace limitations need to be understood in order to avoid faults in the workflow. The proposed robot workflow initializes with a safety-monitored stop during which the robot cart is positioned in front of the patient as directed by the experienced surgeon. A safety-monitored stop is a safety task where the robot motors are physically locked with mechanical brakes, and no motion is possible. Once the robot cart is placed and locked in position, the safety-monitored stop is disabled, and hand-guide mode is initiated. The safety monitoring protocol uses trigger signals send from the main PC hosting the LabView interface through the PCIe-6351 I/O card. Safety-oriented tasks with increasingly more constrained safety parameters are enabled as the application proceeds in order to maintain the risk associated with collaborative robots low. Under hand-guide mode, the surgeon comes into contact with the robot, so a force monitoring safety task is enabled. A safety-monitored stop will activate if the force on any point of the robot exceeds 15N. This number is well under the threshold values for pain as reported by the most recent literature [31] [32]. The initial hand-guide mode is a macro-positioning task under gravity compensation that allows the user to move the robot tool within the imaging workspace, where the OCT imaging system will operate. Macro-positioning is programmed so that the robot holds its position under the weight of the tool and the force of gravity. It does, however, comply to any other external force applied to its axes. with impedance parameters for each robot axis that account for the force of gravity. We refer to such hand-guidance task as macro-positioning because it allows the robot to be moved to any point within its workspace, but precise positioning and orientation of the tool is difficult to achieve. Once the robot is placed within the imaging workspace with macro-positioning, the imaging and aiming lasers are enabled and the program switches to micro-positioning hand-guide mode, a null space positioning task. Micro-positioning, with the help of an aiming laser, is necessary to navigate the OCT system so that the sample is within its field of view (FOV). In contrast to macro-positioning, where each axis is programmed with its own torque parameters, micro-positioning treats the entire robot as a spring-damper model with stiffness and damping parameters. Stiffness and damping parameter are configured to achieve a smooth motion so that delicate position and orientation adjustments can be made by the surgeon. The OCT needs to be positioned so that the sample is within its field of view (FOV). Given the increased proximity to the patient and the more involved operation of the robot by the surgeon, motion commands inside the imaging workspace require the addition of a speed and position monitoring safety mode. After the sample is in the FOV of the imaging system, the surgeon stops all type of hand-guide task and the OCT-based visual servoing takes over in order to compensate for motion in the optical axis. If the sample goes out of the FOV under motion compensation, the robot needs to disable motion to avoid a collision.

The author implemented a risk management procedure in order to validate safety requirements needed for safe robot operation. The safety acceptance review outlined Appendix A, describes the safety configuration that needs to be tested before robot operation. The robot code is presented in Appendix B.

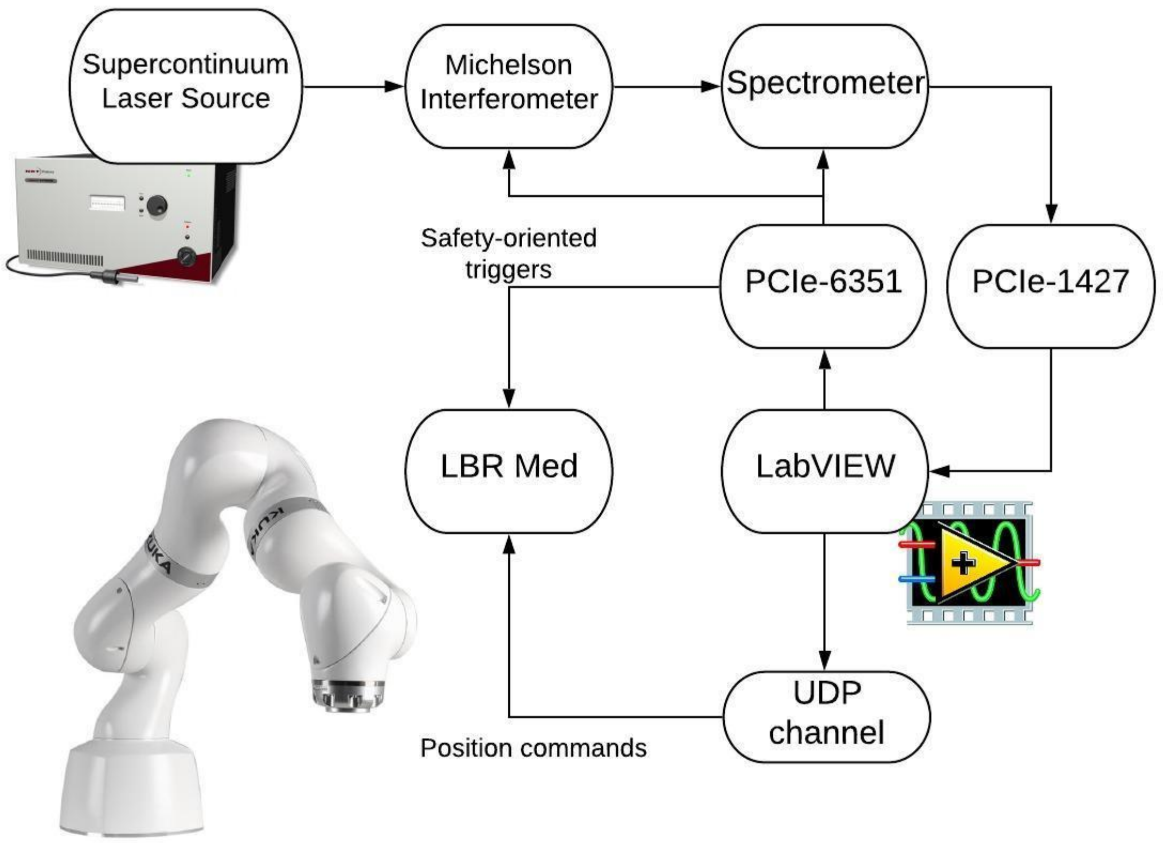
3.2 OCT Integration

Intraoperative imaging has been leveraged by neurosurgeons to better locate brain lesions. The development of collaborative robots has created opportunities for medical tasks to be performed with improved accuracy. To this end, we have developed an intraoperative OCT imaging tool using a collaborative robot to assist the surgeon during brain surgery. System design around collaborative robots emphasize safety. The risks posed by sharing the workspace with a robotic arm are such that redundant systems are necessary to ensure surgeon and patient safety. These redundant systems come in the form of additional sensors and software to monitor the robot and its workspace. Imaging tasks inherently reduce risk because there is no physical contact between the robot and the patient. Despite this, position and speed need to be monitored because close proximity to the patient by the robot still carries risks. Given the collaborative nature of the task, contact with the surgeon by the robot is unavoidable, therefore, force applied by or on the robot needs to be monitored.

Robot-assisted image-guided surgery is giving surgeons options to treat with procedures that were, before, too dangerous for the patient. Intraoperative imaging gives near real-time feedback to the surgeon while robotics allows to guarantee accuracy throughout the surgical procedure. The intraoperative use of OCT has been presented before by a number of research groups. The first projects combining robotics and OCT applied their efforts to retina surgery. Intraoperative OCT applications in neurosurgery have also been explored. A common approach is the use of a hand-held probe to scan the sample. This approach, however, has the drawback of providing an unstable positioning for signal acquisition, which translates to poor image quality. A research group from Germany has demonstrated the use of actuated robotic arms to address these problems [33] [34]. They have integrated an OCT system with a commercial robotized surgical microscope. Their work has revolved around remote-controlled manipulation to position the microscope and the development of an algorithm to register multiple C-scans to display a large volume. A second research project led by J. Izatt is developing an OCT retina alignment system with a robotic arm. This a very involved project where a combination of depth sensors and cameras are used to align and stabilize the retina of the patient for OCT imaging [35]. In contrast, this project demonstrates an OCT system mounted on a collaborative robotic arm navigated by the surgeon and enabled to actively compensate for brain dilation and contractions during surgery. Such imaging platform is not limited to a single repetitive task but becomes useful in surgical procedures where different angles, views, or cross-sections are desired. We have aimed to develop a robotic workflow that is sufficiently flexible to adapt to different surgical workflows as needed by the surgical team.

Vision technology is essential in robotics. The ability to have information about the robot’s environment and react to it by safely changing the motion is necessary to avoid collisions and track objects. Collaborative robot technology has made the robots safer to work with by giving them a sense of touch through torque sensor. However, without robotic vision technology the robot only knows about itself and nothing about its environment. Robotic vision leads us to the question of what the relationship between the robot and the camera is, with respect to the environment where they both operate. Robot motion affects what the camera sees. In the eye-to-hand architecture the vision system is a free-standing system with a macro view of the environment and could potentially be blocked by the robot or other change in the environment, such as a person standing in front of the sensor. With an eye-in-hand architecture or end-point close-loop, in which the camera is mounted on the robot, there is no risk of the robot blocking the camera, but field of view is restricted to what is in front of the robot. This architecture is a robust solution in visual servoing applications as the spatial transformation between imaging system frame and robot base frame is fixed. Workspace restrictions as well as coordinate system relationships need to be understood in order integrate robotic vision systems. Image-based visual servoing under the eye-in-hand architecture requires the relationship between camera and robot coordinate system to relate the desired camera motion to the required robot motion. Pose estimation is left to the robot controller and a relative position command is transferred from the tracking controller in Cartesian space. A schematic block diagram of the system integration, Fig 23, shows LabView used to synchronize the scanning-galvo and the spectrometer, acquire and process OCT data, send safety-oriented trigger signals to the robot, implement proportional control, and finally send the output of the controller through a UDP channel to the LBR Med. An Ethernet-based UDP communication protocol is used to send motion commands to the robot controller from the PC hosting the OCT imaging software and controller. Motion command transfer is done at a rate of 50Hz, limited by the computational speed of OCT imaging. OCT imaging is data intensive and computationally expensive. We found limitations related to imaging data processing speed and optic signal stability. The LabView control panel for these tasks is shown in Fig 24a. The frame transformation diagram for the LBR Med and OCT end-effector and its mounting on the robot is shown in Fig 24 b&c.

[a]



[b]

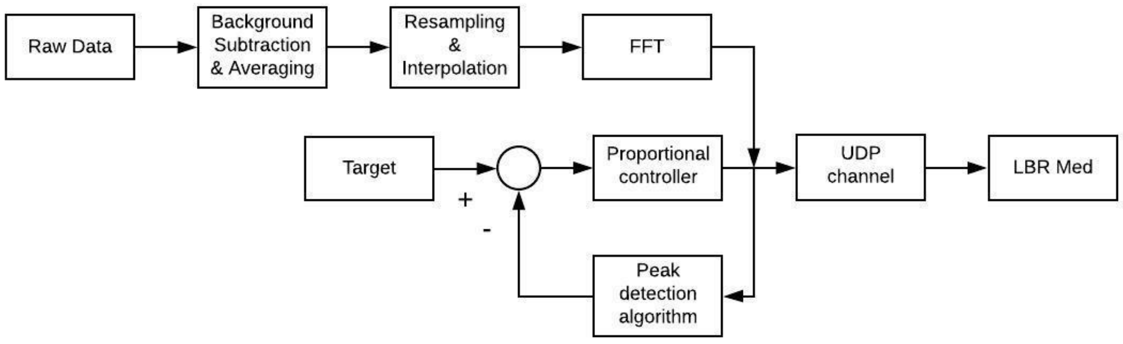
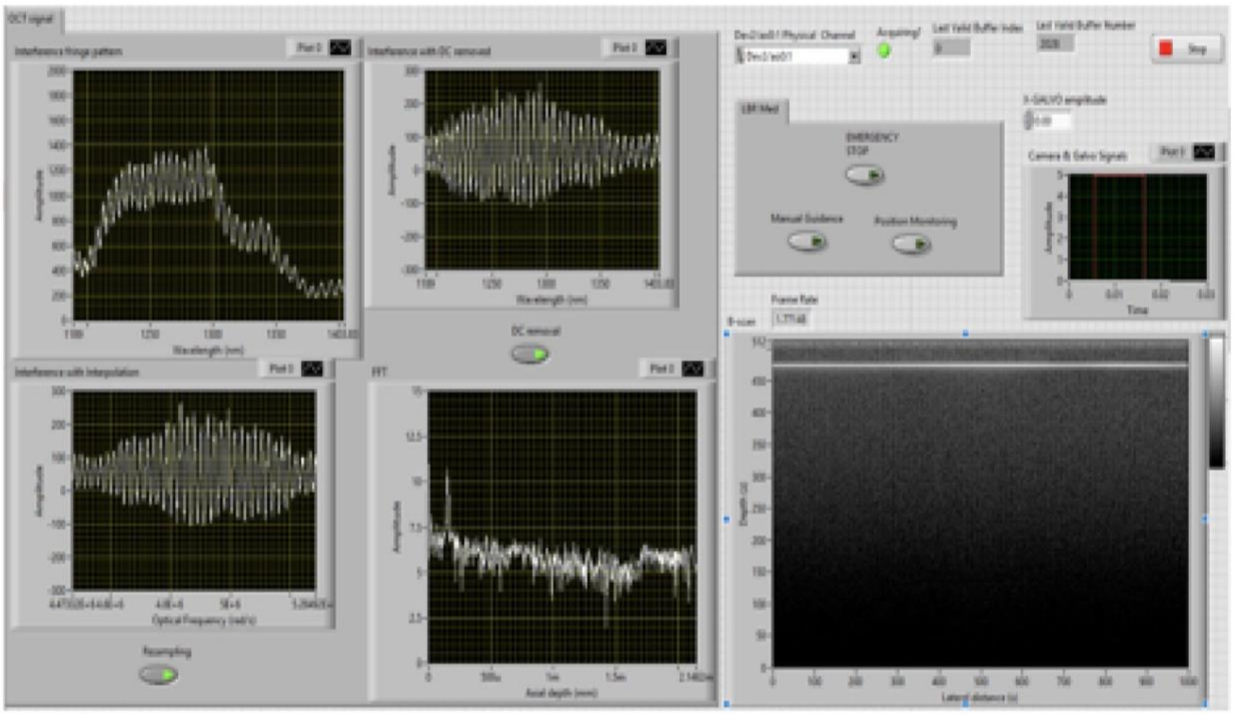
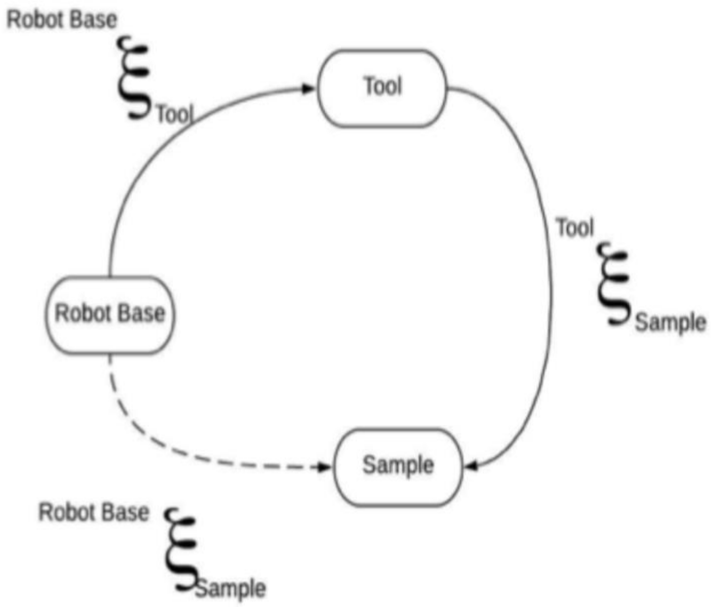


Figure 23. [a] System integration block diagram. [b] Data flow block diagram.

[a]



[b]



[c]

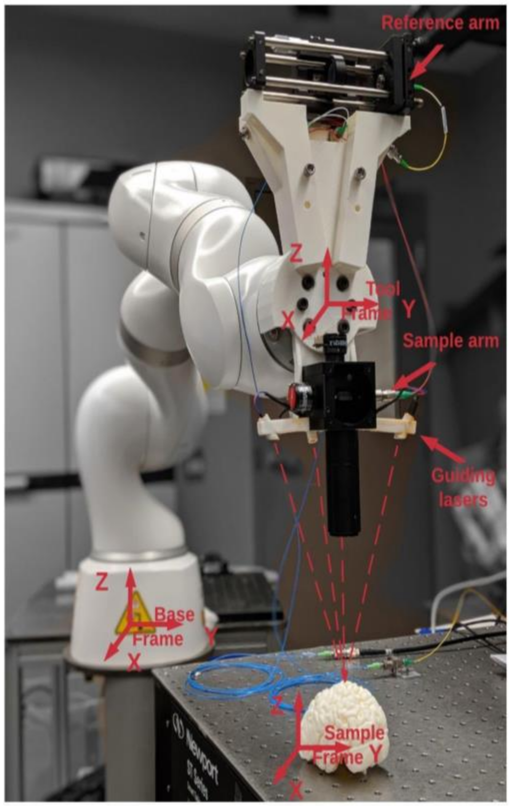


Figure 24. [a] LabView GUI for the LBR_OCT system. [b] Frame transformation diagram. [c] OCT end-effector with aiming/guiding lasers and interferometer.

3.3 Experiment methodology and Results

The experiment set up uses a galvo motor and a test sample with high-reflectivity, Fig. The galvo motor is controlled through LabView where square, sine, and ICP waveforms can be easily generated. The test sample has a simple intensity profile which simplifies the peak detection solution needed. This allows us to easily observe the controller response without concerns on the peak detection algorithm.

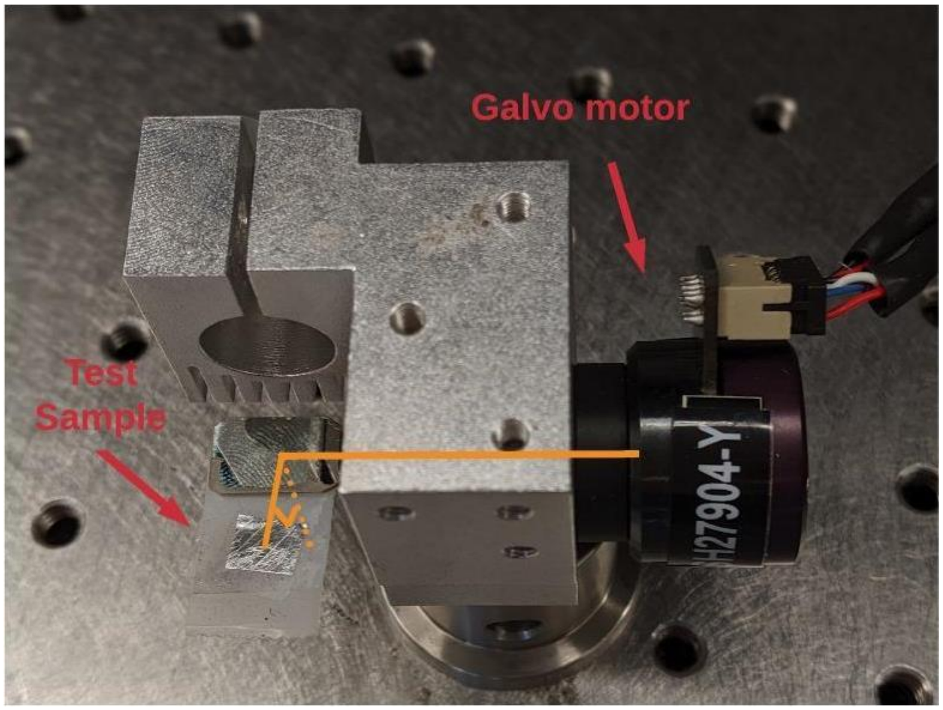


Figure 25. Galvo motor and test sample.

Intracranial Pressure (ICP) Waveform and Proportional Controller

Intracranial pressure (ICP) pulsation is caused primarily by cerebrospinal fluid (CSF) flow. Cerebrospinal fluid (CSF) contained in the brain ventricles and between the cerebral cortex and brain meninges serves to protect and regulate neural functioning. CSF circulation is pulsatile and in correlation with the cardiac cycle and respiration rate [36][37]. The simplest model of ICP pulsations consists of a sinusoidal waveform, however, the true morphology of the waveform is dictated by complex physiologic events. In this experiment, we perform our oscillation using an ICP pulse train generated using the model provided by Wadehn [38]. The waveform generated by this model, shown in Fig 25, uses the Ursino-Lodi model of ICP dynamics [39]. The morphology of the signal has three peaks called P1 (wave originating from systolic arterial blood pressure pulsations), P2 (CSF pressure wave) and P3 (due to venous pulsations). Respiration induced changes in the cardiovascular system modulates ICP with a sinusoid wave whose frequency is the respiratory rate. The frequency domain signal shows the slow ICP dynamics modelled by Ursino-Lodi, the respiration rate and the cardiac-induced pulse with its harmonics. The ICP signal is obtained by adding all three components.

$$ICP(t) \; = \; ICP_{CSF}(t) + ICP_{cardiac}(t) + ICP_{resp}(t)$$

Equation [18]. Intracranial Pressure (ICP) waveform composition.

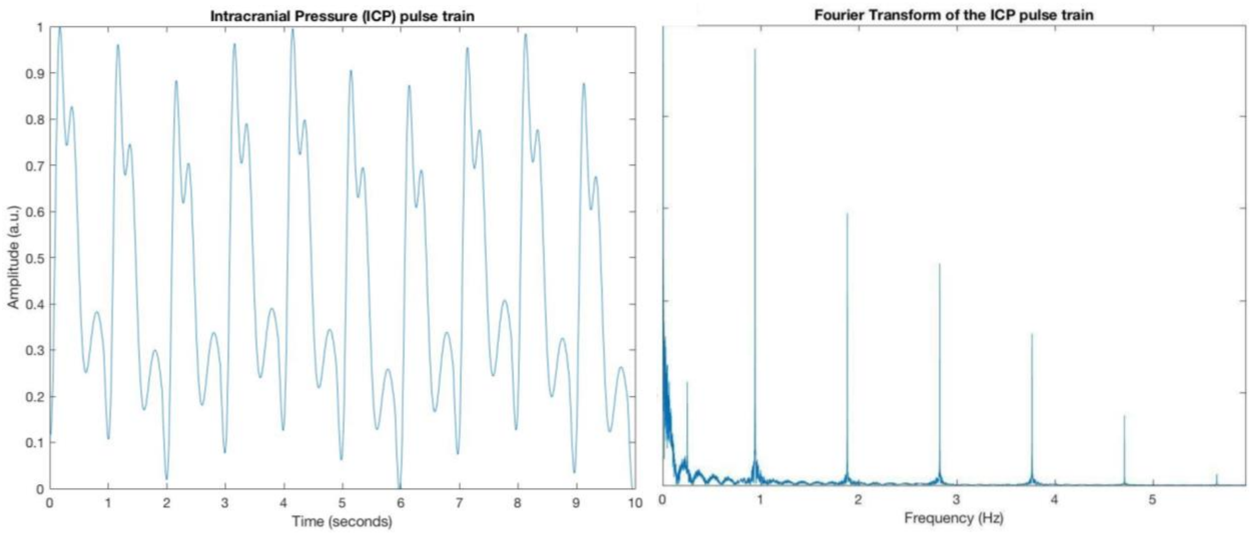


Figure 26. Intracranial Pressure (ICP) waveform and frequency spectrum.

We implement a proportional control loop which allows the robot to compensate for sample motion in the direction of the optic axis (z-axis). It is important to understand the kinematics of the problem as defined by the relationship between robot, camera, and sample space. As described in Fig 24, the robot base frame serves as the world coordinate system from which every other frame is derived. The robot base, tool, and sample frames have been defined so that their axes have the same orientation in order to simplify spatial transformations. We use the proportional controller to maintain the peak intensity position in the target location. The peak intensity detection operation on an OCT A-line gives the location of the surface of the sample. The location of the peak is continuously computed, and the controller maintains it at a target level, within the FOV of the imaging system, by sending relative displacement commands to the robot. The control problem is solved in image space, in terms of pixel number, but the relative position commanded of the robot needs to be sent in meters (m). Given the probing depth value presented in the OCT imaging chapter ($z_{max} = 2.1mm$) and the number of pixels in the direction of the z-axis (512 pixels), we can calculate the size of a pixel in the real world. A pixel in space domain represents ~4μm. After defining how we want points to move in the image the robot to move, we construct a controller that moves the camera so as to create the desired point motion. The proportional controller employs feedback to calculate the error, $e(t)$, between a required value, $r(t)$, and the current value, $y(t)$, of a measured variable. The controller calculates the current value of the error function, $e(t)$, to drive the measured variable to the required setpoint. The proportional term, K_p, determines the contribution the error value will have on the controller response. The proportional term weights on the current error value so that a large K_p will drive the error function to zero faster than a small K_p value. The use of a large proportional term, however, will cause the controller to overshoot the desired setpoint, or it will become unstable.

$$e(t) = [r(t) - y(t)] * 4 * 10^{-6}$$

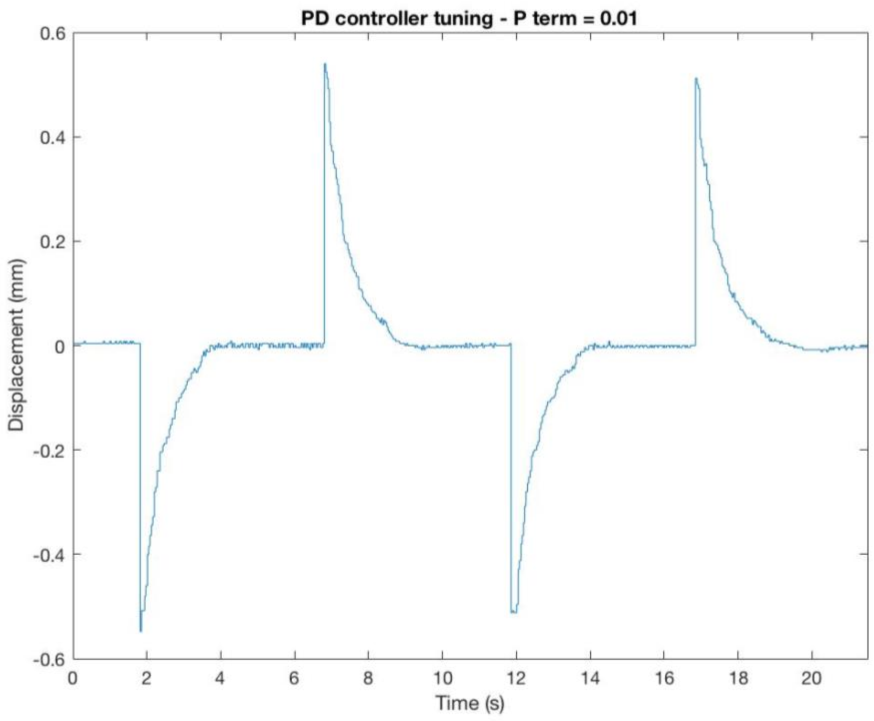
$$u(t) = K_p e(t)$$

Equation [19]. Proportional controller function.

Given the controller type, we have elected to manually tune the coefficient term. A square wave is generated to drive the galvo motor and obtain the step response. The step response is used to observe the time evolution response of the controller to a sudden change in input. The step response of the controller is shown in Fig 27 as the K_p variable increases and it is used to calculate rise time, overshoot, and settling time until the desired response is obtained. The rise time is the time taken by the controller to reach the desired value when there is an instantaneous step input. The overshoot is a percentage ratio between the desired target and the maximum value reached by the controller. The settling time is taken as the time needed to reach a stable value after the step input has been applied.

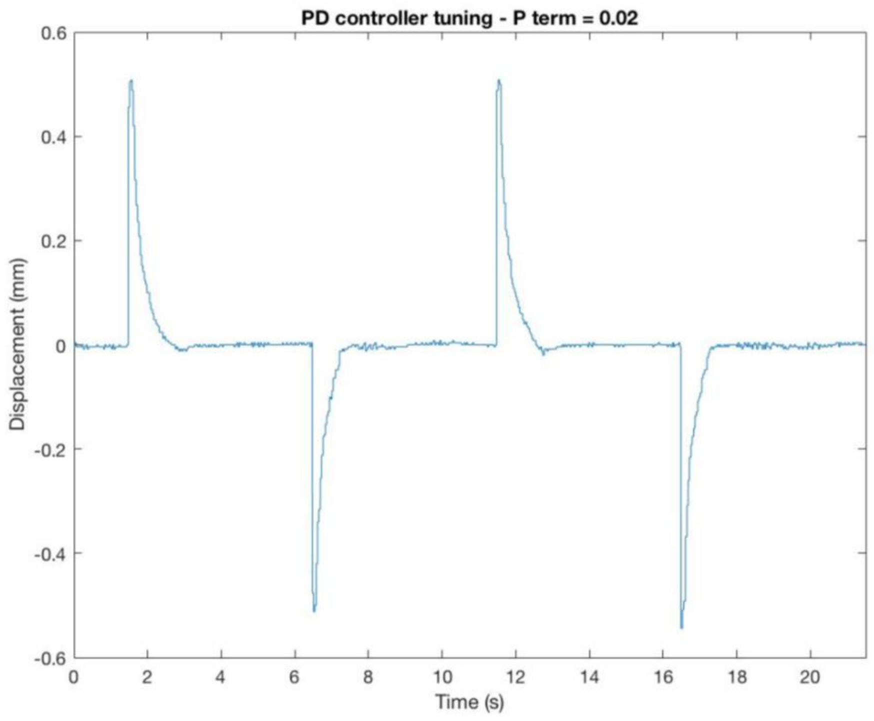
The goal of the controller is to track the ICP waveform. The frequency spectrum of the ICP signal has low frequencies (<1Hz) and a main frequency component of 1Hz with its harmonics. We generate a series of sinusoidal waves at the desired frequencies to serve as the drive signal to the galvo motor. Fig 28a-c show the drive signal in blue and the tracking error in orange. Fig 28d shows the magnitude frequency response of the controller with a stable tracking error for the desired frequency band. Lastly, Fig 29 shows the ICP waveform generated at a frequency of 1Hz, shown in blue. The controller is able to track the motion with an average peak error of 0.31mm, shown in orange, as expected from the frequency response.

[a]



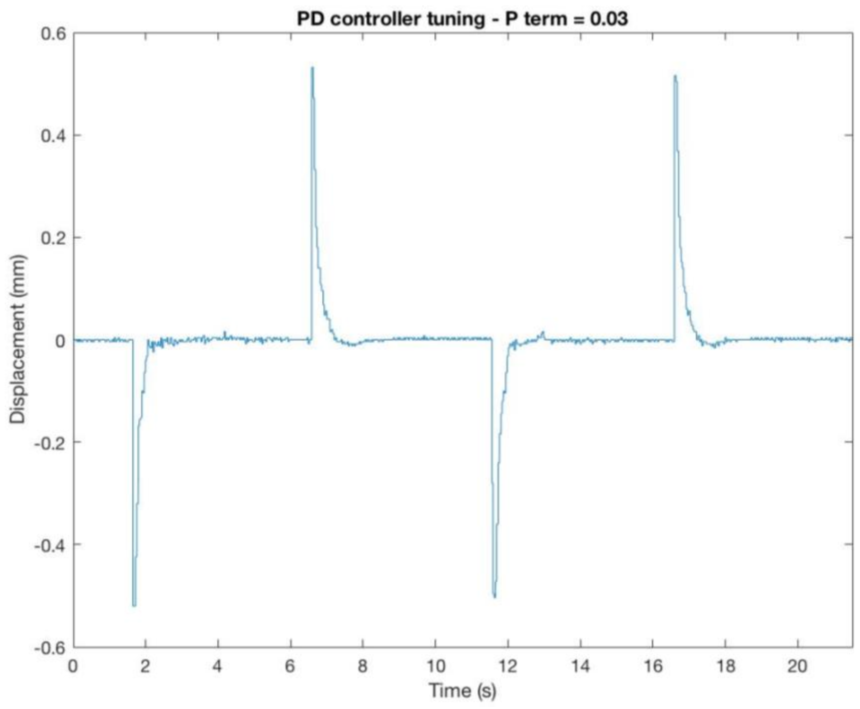
Rise time = 1.3s Overshoot = 0% Settling time = 2.32s

[b]



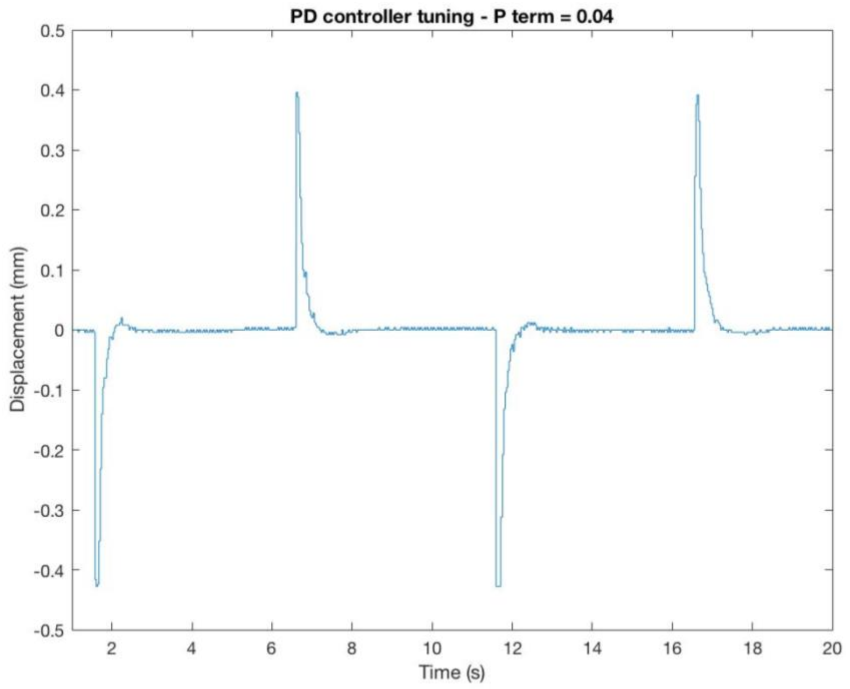
Rise time = 0.62s Overshoot = 0.9% Settling time = 1.59s

[c]



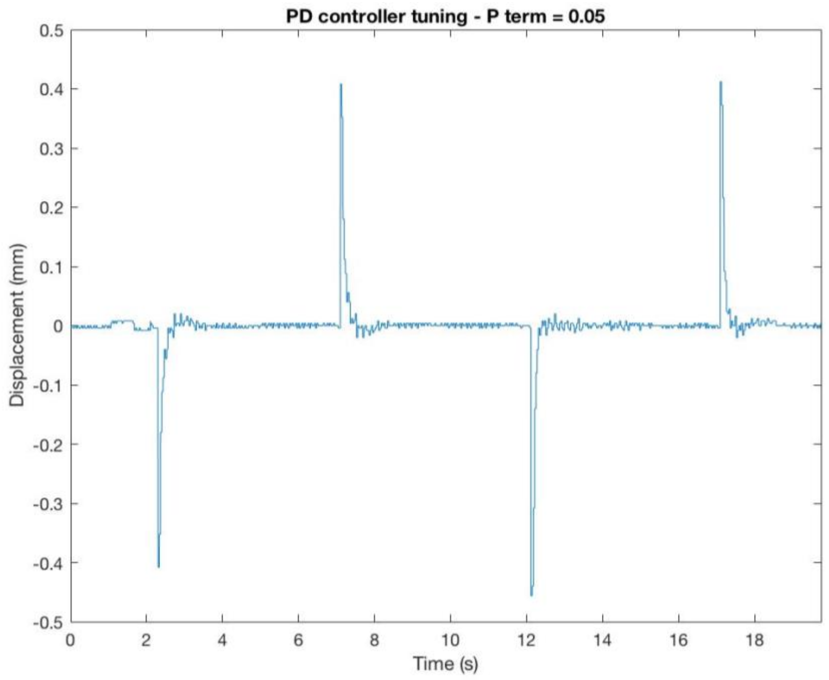
Rise time = 0.36s Overshoot = 1.2% Settling time = 1.51s

[d]



Rise time = 272ms Overshoot = 4.2% Settling time = 1.38s

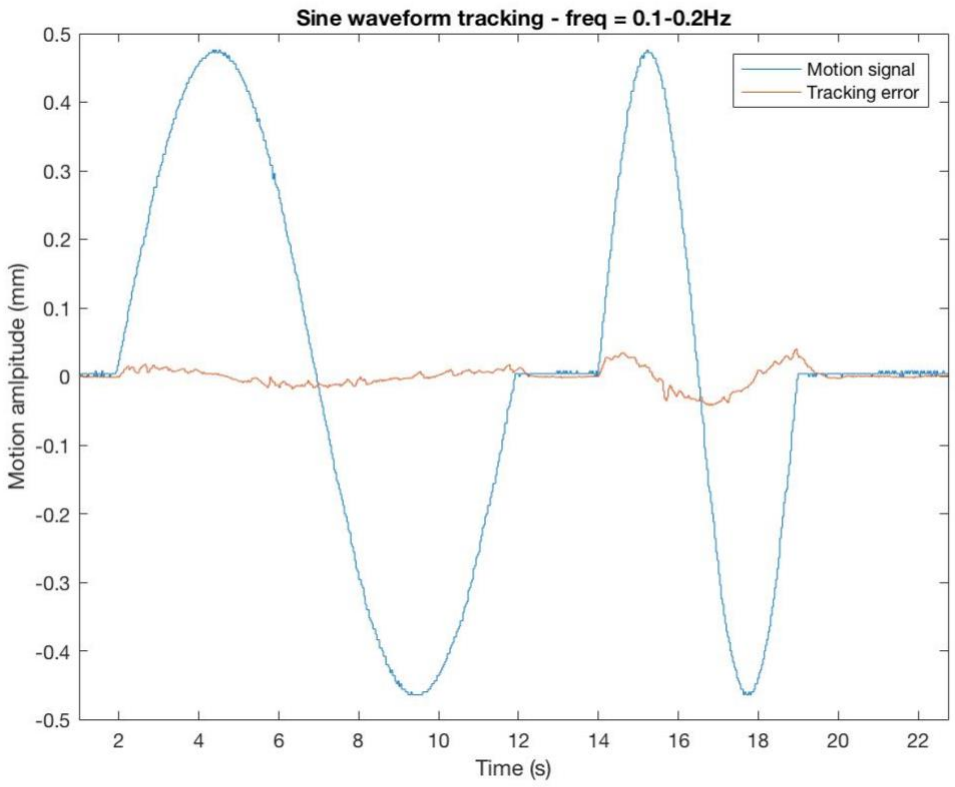
[e]



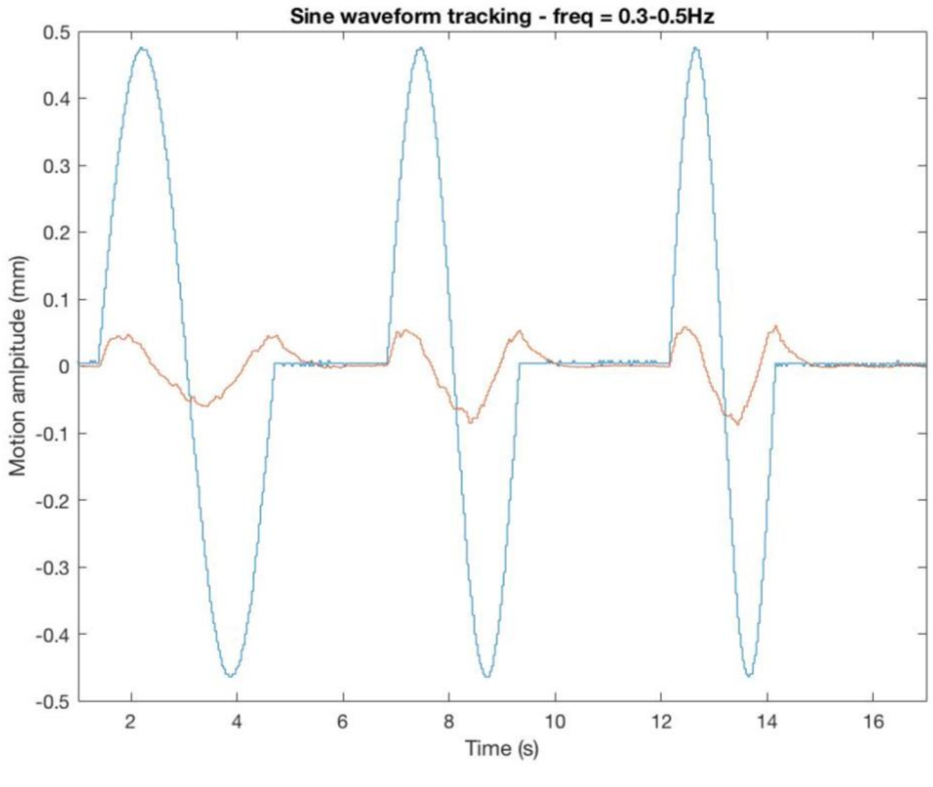
Rise time = 171ms Overshoot = 4.4% Settling time = 1.27s

Figure 27. Proportional controller tuning.

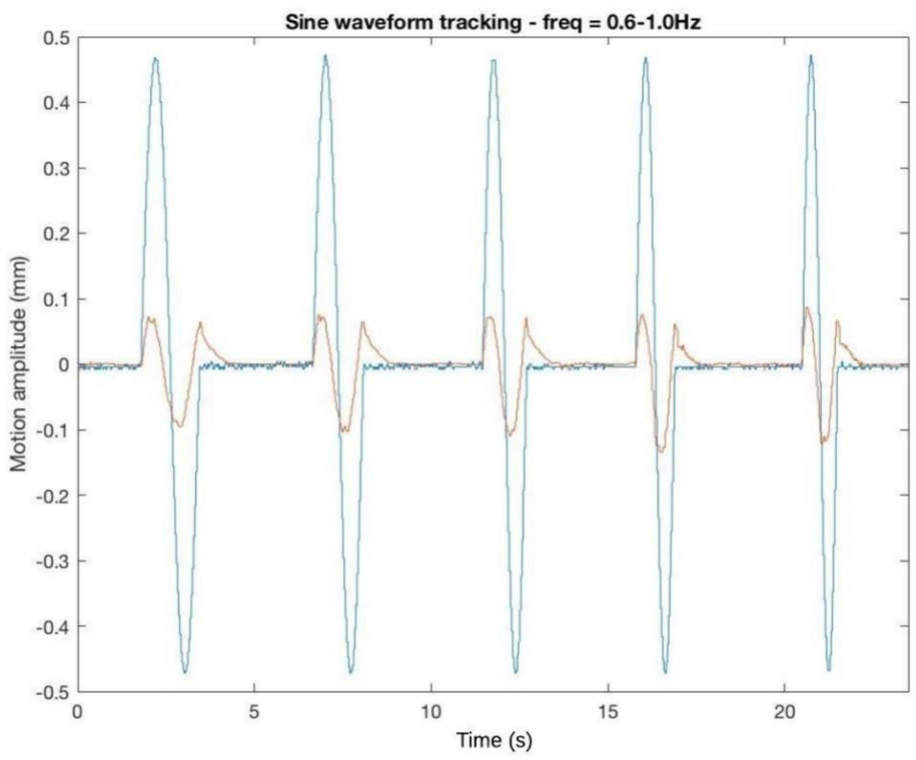
[a]


Sine waveform tracking - freq = 0.1-0.2Hz
Motion signal
Tracking error
Motion amlpitude (mm)
Time (s)
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
2
4
6
8
10
12
14
16
18
20
22

[b]


Sine waveform tracking - freq = 0.3-0.5Hz
Motion amlpitude (mm)
Time (s)
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
2
4
6
8
10
12
14
16

[c]



[d]

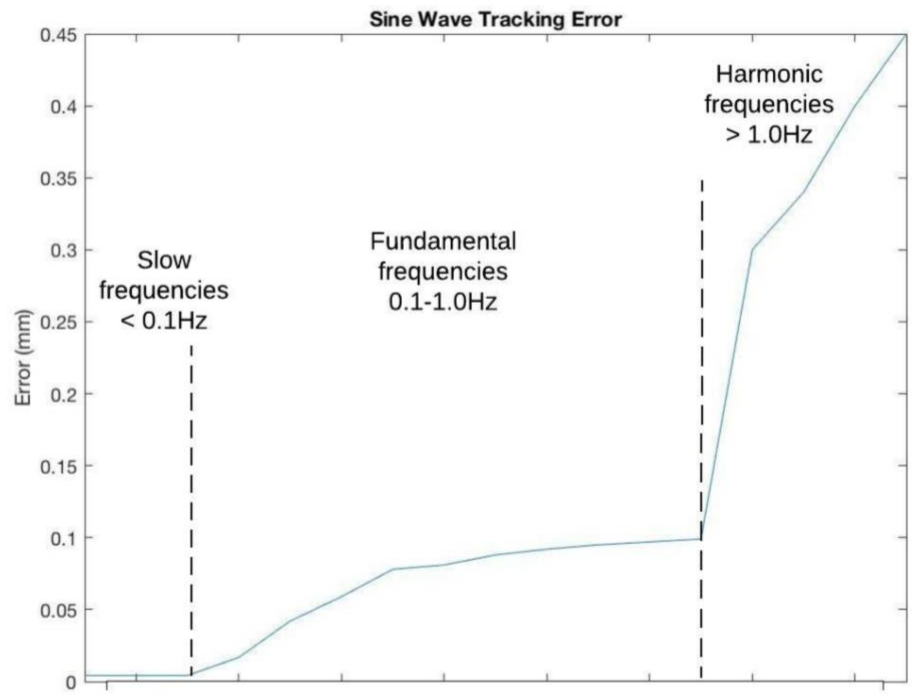


Figure 28. Frequency response of the proportional controller.

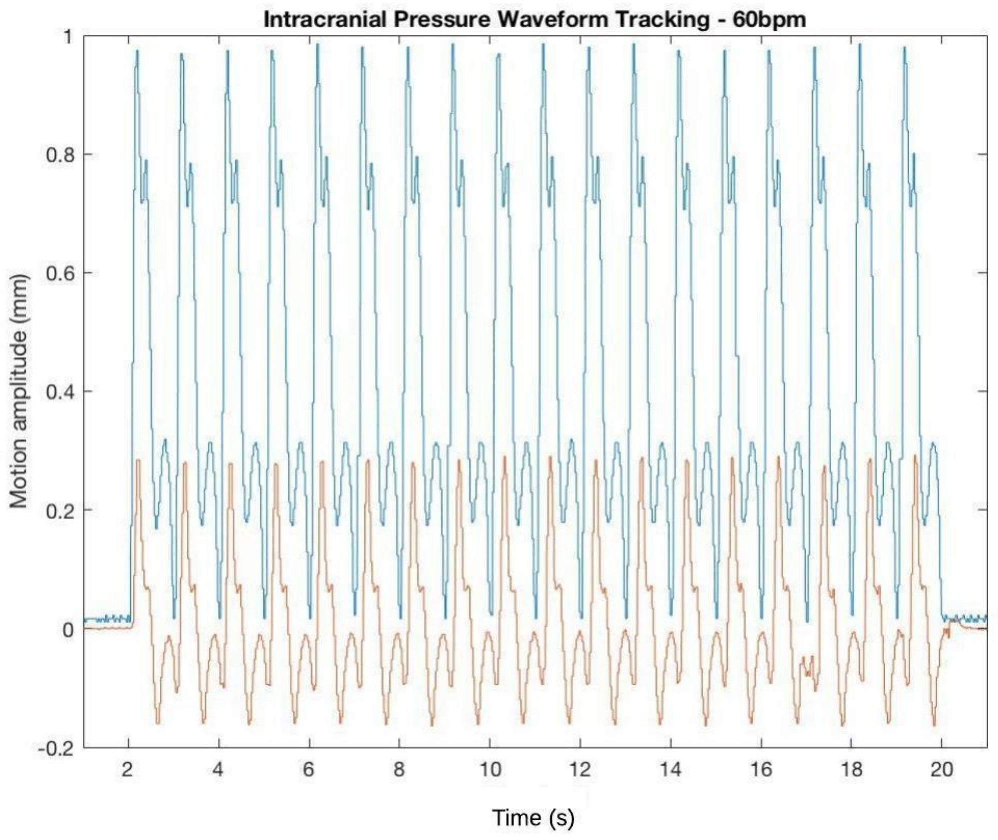


Figure 29. Intracranial pressure waveform signal and tracking error

4. Conclusion

The work presented in this thesis proposes a solution to brain motion compensation for OCT imaging using a collaborative robot arm. Advantages and disadvantages of such tool can be understood, from the surgeon's perspective, to be a trade-off between ease of use and clinical outcome improvement. The outcome of this project was successful as it demonstrated how surgeon and robot can safely collaborate to perform an imaging task. However, further engineering development is needed to make this system into a fully operational tool ready to be deployed in the operating room.

A major disadvantage with OCT-based robot navigation is that OCT imaging is a data intensive modality. Acquisition and processing of large volumetric OCT data is slow and computationally expensive. Optimized hardware solutions (i.e. GPU, FPGA) are needed to reduce the computational overhead. The current probing depth of the imaging system is relatively shallow at ~2mm. Deeper tissue penetration can be obtained through hardware by swapping the spectrometer's camera for a model with more elements in the sensor array.

The test sample used in the experiment can be upgrade to a brain phantom in order to create a more realistic intensity profile to develop a better peak intensity detection algorithm. In addition, a patient under anesthesia has a cardiac rate higher than 60 beats/second so a faster controller response is desired in a real setting.

In conclusion, intraoperative imaging with robot collaboration is feasible. An experienced surgical staff can leverage the technology to ease workflow and improve clinical outcome. Safety software and hardware protocols ensure that fault scenarios are handled in the safest way possible.

This has given way to new use cases for robots in surgery. The LBR Med is the result of decades of research in collaborative robots and its currently the only collaborative robot arm certified to be integrated in a medical device. The demonstration of our system shows the possibilities for novel imaging devices with collaborative robots in a surgical setting.

Appendix A: Safety acceptance overview – System safety functions

No.	Activity	Yes	No
1	Operator safety: is all operator safety equipment configured, properly connected and tested for correct function?	✓	
2	Operator safety: a stop is triggered if AUT or T2 mode is active with the operator safety open.	✓	
3	Operator safety: a manual reset function is present and activated.	✓	
4	Brake test: is a brake test planned and has an application been created for this purpose?	✓	
5	Hand guiding device enabling state: is the enabling device of the hand guiding device configured, properly connected and tested for correct function?	✓	
6	Local EMERGENCY STOP: are all local EMERGENCY STOP devices configured, properly connected and tested for correct function?	✓	
7	External EMERGENCY STOP: are all external EMERGENCY STOP devices configured, properly connected and tested for correct function?	✓	
8	Local and external EMERGENCY STOP: are the local and external EMERGENCY STOPs each configured as an individual AMF in a row of the PSM table?	✓	
9	Safety stop: is all operator safety equipment configured, properly connected and tested for correct function?	✓	
10	Safe operational stop: is all equipment for the safe operational stop configured, properly connected and tested for correct function?	✓	
11	When using position-based AMFs: is the limited safety integrity of the position-based AMFs taken into consideration in the absence of position referencing?	✓	
12	When using position-based AMFs: has position referencing been carried out successfully?	✓	
13	If external position referencing is used: has a suitable test method for position mastering been provided?	✓	
14	If external position referencing is used: has it been ensured that the input is only set after successful testing?	✓	
15	If external position referencing is used: has it been ensured that the input is only set after successful testing?	✓	
16	Manual guidance: has it been configured in such a way that	✓	

	appropriate velocity monitoring is active in every operating mode for manual guidance?		
17	If using the enabling device of the hand guiding device as an input for deactivating safety functions: Has it been taken into consideration that using the enabling device as an input may result in safety functions being deactivated during manual guidance?	✓	
18	Collision detection: have all necessary HRC functionalities been configured?	✓	
19	Collision detection: has it been configured in such a way that velocity monitoring is also always active when collision detection is active?	✓	
20	Collision detection: has it been configured in such a way that velocity monitoring is also always active when TCP force monitoring or monitoring of a base-related TCP force component is active?	✓	
21	Collision detection: is a safety stop 0 configured for all safety monitoring functions in order to detect crushing situations?	✓	
22	When using axis torque-based AMFs: is the limited safety integrity of the axis torque-based AMFs taken into consideration in the absence of position referencing and/or torque referencing?	✓	
23	In the configuration of all rows in the PSM table and all ESM states, has it been taken into account that the safe state of the AMFs is the “violated” state (state “0”)?	✓	
24	PSM configuration: in the configuration of output signals, has it been taken into account for the safety reaction that an output is LOW (state “0”) in the safe state?	✓	
25	PSM configuration: Was a check carried out during configuration of the “Brake” safety reaction to see whether there could be an increased risk due to rapid switching to and from the violation state of the AMFs with which the Cartesian velocity monitoring is linked?	✓	
26	ESM configuration: are all ESM states consistent, i.e. does each individual ESM state sufficiently reduce all dangers?	✓	
27	Have torque and position referencing been carried out successfully?	✓	
Signed: Robnier Reyes Perez			

Appendix B: LBR Med Code

```
package com.kuka.connectivity.fri.example;

import static com.kuka.roboticsAPI.motionModel.BasicMotions.positionHold;
import static com.kuka.roboticsAPI.motionModel.BasicMotions.ptp;

import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import com.kuka.connectivity.fastRobotInterface.FRIConfiguration;
import com.kuka.connectivity.fastRobotInterface.FRIJointOverlay;
import com.kuka.connectivity.fastRobotInterface.FRISession;
import com.kuka.roboticsAPI.applicationModel.RoboticsAPIApplication;
import com.kuka.roboticsAPI.controllerModel.Controller;
import com.kuka.roboticsAPI.deviceModel.LBR;
import com.kuka.roboticsAPI.geometricModel.CartDOF;
import
com.kuka.roboticsAPI.motionModel.controlModeModel.CartesianImpedanceControlMode;
import com.kuka.roboticsAPI.motionModel.controlModeModel.PositionControlMode;
import com.kuka.roboticsAPI.sensorModel.DataRecorder;
import com.kuka.roboticsAPI.sensorModel.DataRecorder.AngleUnit;
import com.kuka.roboticsAPI.uiModel.ApplicationDialogType;

/**
 * Creates a FRI Session.
 */
public class LBRJointSineOverlay extends RoboticsAPIApplication
{
   private Controller _lbrController;
   private LBR _lbr;
   private String _clientName;

   @Override
   public void initialize()
   {
      _lbrController = (Controller) getContext().getControllers().toArray()[0];
      _lbr = (LBR) _lbrController.getDevices().toArray()[0];
      // **********************************************************************
      // *** change next line to the FRIClient's IP address                ***
      // **********************************************************************
      _clientName = "192.170.10.5";
   }
```

```
    @Override
    public void run()
    {
        DataRecorder rec1 = new DataRecorder("J6data.log", 25, TimeUnit.SECONDS, 1);
        rec1.addCurrentJointPosition(_lbr, AngleUnit.Radian);
        rec1.addCommandedJointPosition(_lbr, AngleUnit.Radian);
            //Position Control Mode Configuration
            PositionControlMode posHold = new PositionControlMode();
            //Impedance Control Mode Configuration
            CartesianImpedanceControlMode impModeX = new CartesianImpedanceControlMode();
                    impModeX.parametrize(CartDOF.X).setStiffness(200).setDamping(0.1);

                    CartesianImpedanceControlMode       impModeTRANSL       =       new
CartesianImpedanceControlMode();

            impModeTRANSL.parametrize(CartDOF.TRANSL).setStiffness(100).setDamping(0.1);

                    CartesianImpedanceControlMode       impModeROT       =       new
CartesianImpedanceControlMode();
                    impModeROT.parametrize(CartDOF.ROT).setStiffness(50).setDamping(0.1);

                    CartesianImpedanceControlMode       impModeALL       =       new
CartesianImpedanceControlMode();
                    impModeALL.parametrize(CartDOF.ALL).setStiffness(50).setDamping(0.1);

                    CartesianImpedanceControlMode       impModeXSweep       =       new
CartesianImpedanceControlMode();
                    impModeXSweep.parametrize(CartDOF.Y).setStiffness(200).setDamping(0.1);
                    impModeXSweep.parametrize(CartDOF.A).setStiffness(0.1).setDamping(0.1);
                    CartesianImpedanceControlMode       impModeYSweep       =       new
CartesianImpedanceControlMode();
                    impModeYSweep.parametrize(CartDOF.Z).setStiffness(200).setDamping(0.1);
                    impModeYSweep.parametrize(CartDOF.B).setStiffness(0.1).setDamping(0.1);

                     // move to start pose
        //_lbr.move(ptp(Math.toRadians(16.93),     Math.toRadians(48.76),     Math.toRadians(0),
Math.toRadians(-78.49), Math.toRadians(0.79), Math.toRadians(-39.22), Math.toRadians(-1.23))
            //      .setJointVelocityRel(0.2));
                    _lbr.move(positionHold(posHold, 1,TimeUnit.SECONDS));
            // configure and start FRI session
        FRIConfiguration  friConfiguration  =  FRIConfiguration.createRemoteConfiguration(_lbr,
_clientName);
        friConfiguration.setSendPeriodMilliSec(1);


        getLogger().info("Creating FRI connection to " + friConfiguration.getHostName());
        getLogger().info("SendPeriod: " + friConfiguration.getSendPeriodMilliSec() + "ms |"
                + " ReceiveMultiplier: " + friConfiguration.getReceiveMultiplier());
        FRISession friSession = new FRISession(friConfiguration);
        FRIJointOverlay jointOverlay = new FRIJointOverlay(friSession);

        // wait until FRI session is ready to switch to command mode
        try
        {
           friSession.await(10, TimeUnit.SECONDS);
        }
        catch (final TimeoutException e)
        {
           getLogger().error(e.getLocalizedMessage());
           friSession.close();
           return;
        }

        getLogger().info("FRI connection established.");

        int option0 = 0;
        option0   =   getApplicationUI().displayModalDialog(ApplicationDialogType.QUESTION,
"What's next?", "Hold position", "Imaging position");
           switch(option0){
                   case 0:
                           getLogger().info("Hold position");
                           _lbr.move(positionHold(posHold, 1,TimeUnit.SECONDS));
                           break;
                   case 1:
                           getLogger().info("Moving to Imaging position");
                           _lbr.move(ptp(Math.toRadians(0),                    Math.toRadians(45),
Math.toRadians(0),      Math.toRadians(-75),      Math.toRadians(0),      Math.toRadians(-30),
Math.toRadians(0))
                           .setJointVelocityRel(0.2));
                           break;


        }

        int option1 = 0;
                   int option2 = 0;
                   while (option1!=7){
                   option1                                                                   =
getApplicationUI().displayModalDialog(ApplicationDialogType.QUESTION,  "What's  next?",
"Macro-positioning",   "Micro-positioning","'Soft'   Mode",   "Trocar   Guiding","OCT   X
Sweep","OCT Y Sweep", "Brain Motion Compensation", "Finish");
                   switch(option1){
                   case 0:
                           /*_lbr.setESMState("2");
                      _lbr.move(handGuiding());
```

```
                    _lbr.setESMState("1");
                    _lbr.move(positionHold(posHold, 1,TimeUnit.SECONDS));*/
                        getLogger().info("HandGuide Mode");
                        _lbr.move(positionHold(posHold, 1,TimeUnit.SECONDS));
                    break;
                  case 1:
                        getLogger().info("Micro-positioning");
                        //_lbr.setESMState("3");
                        while(option2!=3){
                              option2                                    =
getApplicationUI().displayModalDialog(ApplicationDialogType.QUESTION,          "Micro-
Position?", "TRANSL", "ROT","ALL", "Finish");
                              switch(option2){
                              case 0:
                                    _lbr.move(positionHold(posHold,
1,TimeUnit.SECONDS));
                                    _lbr.move(positionHold(impModeTRANSL,
1,TimeUnit.SECONDS));
                                    break;
                              case 1:
                                    _lbr.move(positionHold(posHold,
1,TimeUnit.SECONDS));
                                    _lbr.move(positionHold(impModeROT,
1,TimeUnit.SECONDS));
                                    break;
                              case 2:
                                    _lbr.move(positionHold(posHold,
1,TimeUnit.SECONDS));
                                    _lbr.move(positionHold(impModeALL,
1,TimeUnit.SECONDS));
                                    break;
                              case 3:
                                    _lbr.move(positionHold(posHold,
1,TimeUnit.SECONDS));
                                    break;
                              }
                        }
                    //_lbr.setESMState("1");
                    _lbr.move(positionHold(posHold, 1,TimeUnit.SECONDS));
                break;
                  case 2:
                        getLogger().info("Soft mode");
                        //_lbr.setESMState("3");
                    _lbr.move(positionHold(impModeALL, 30,TimeUnit.SECONDS));
                  case 3:
                        _lbr.move(positionHold(posHold, 1,TimeUnit.SECONDS));
                        _lbr.move(positionHold(impModeX, 1,TimeUnit.SECONDS));
                        break;
                        case 4:
                                _lbr.move(positionHold(posHold, 1,TimeUnit.SECONDS));
                                //oct_tool.getFrame("OCT_TCP").move(positionHold(impModeXSweep,
1,TimeUnit.SECONDS));
                                break;
                        case 5:
                                _lbr.move(positionHold(posHold, 1,TimeUnit.SECONDS));
                                //oct_tool.getFrame("OCT_TCP").move(positionHold(impModeYSweep,
1,TimeUnit.SECONDS));
                                break;
                        case 6:
                                getLogger().info("Brain Motion Compensation");
                                rec1.enable();
                                rec1.startRecording();
                                _lbr.move(positionHold(posHold,25,TimeUnit.SECONDS)
                           .addMotionOverlay(jointOverlay));
                                rec1.stopRecording();
                break;
                        case 7:
                                //_lbr.setESMState("1");
                _lbr.move(positionHold(posHold, 1,TimeUnit.SECONDS));
                break;
                        }}

                        /*

                        lbr.setESMState("1");
                        lbr.move(positionHold(posHold, 1,TimeUnit.SECONDS));
                        */

        // move to start pose
        //_lbr.move(ptp(Math.toRadians(19.5),    Math.toRadians(46.36),    Math.toRadians(0.01),
Math.toRadians(-85.53),   Math.toRadians(26.54),   Math.toRadians(-46.74),   Math.toRadians(-
19.23))
          //        .setJointVelocityRel(0.2));

        // async move with overlay ...
       // _lbr.move(ptp(0, Math.toRadians(45), 0, Math.toRadians(-75), 0, Math.toRadians(-30), 0)
       //        .setJointVelocityRel(0.2)
       //        .setBlendingRel(0.1)
        //       );
        //_lbr.move(positionHold(posHold,10,TimeUnit.SECONDS)
          //   .addMotionOverlay(jointOverlay)
         //    );

       /* // ... blending into sync move with overlay
        _lbr.move(ptp(0, Math.toRadians(45), 0, Math.toRadians(-75), 0, Math.toRadians(-30), 0)
             .setJointVelocityRel(0.2)
```

```
            .addMotionOverlay(jointOverlay)
            );*/

        // done
        friSession.close();
    }

    /**
     * main.
     *
     * @param args
     *            args
     */
    public static void main(final String[] args)
    {
        final LBRJointSineOverlay app = new LBRJointSineOverlay();
        app.runApplication();
    }

}
```

Appendix C:

OCT code

/************************* LBROverlayApp.cpp ****************************/***

```
#include <cstdlib>
#include <cstdio>
#include <cstring>
#include "LBRJointSineOverlayClient.h"
#include "friUdpConnection.h"
#include "friClientApplication.h"
#include <NIDAQmx.h>
#include <niimaq.h>
#include <windows.h>
#include <mmsystem.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <wincon.h>
#include "HLGrab.h"
#include <thread>
#include <fftw3.h>
#include <boost/math/interpolators/cubic_b_spline.hpp>
#define _NIWIN

struct mainType {
	int argc;
	char *argv[];
};

int FRIMAIN(struct mainType *FRIMAINStruct);
void sendToClient(LBRJointSineOverlayClient *function);

//THINGS FOR THE DISPLAY START-----------------------------------------------------------------
-
#define DAQmxErrChk(functionCall) if( DAQmxFailed(error=(functionCall)) ) goto Error; else

// error checking macro
#define errChk(fCall) if (error = (fCall), error < 0) {goto Error;} else

#define PI	3.1415926535
#define Cspeed = 299792458;

int32 CVICALLBACK DoneCallback(TaskHandle taskHandle, int32 status, void *callbackData);


// Window proc
```

```
LRESULT CALLBACK ImaqSmplProc(HWND hWnd, UINT iMessage, UINT wParam, LONG
lParam);
// Error display function
void DisplayIMAQError(Int32 error);
// Callbacks
int OnGrab(void);
DWORD ImaqThread(LPDWORD lpdwParam);
DWORD StopThread(LPDWORD lpdwParam);
DWORD GalvoThread(LPDWORD lpdwParam);
DWORD TriggerThread(LPDWORD lpdwParam);
// Thread objects
static HANDLE HThread;
static HANDLE HGalvoThread, HTriggerThread;
static HANDLE HStopThread, HStopEvent;

// windows GUI globals
static HINSTANCE    hInst;
static HWND                 ImaqSmplHwnd;
static HWND         HStop, HGrab, HQuit, HIntfName, HFrameRate;
static HWND                         HPeakPosition;


// Imaq globals
static SESSION_ID   Sid = 0;
static BUFLIST_ID   Bid = 0;
static INTERFACE_ID Iid = 0;
static uInt16         *ImaqBuffer = NULL;    // acquisiton buffer
static Int32        CanvasWidth = 512;  // width of the display area
static Int32        CanvasHeight = 512; // height of the display area
static Int32        CanvasTop = 10;     // top of the display area
static Int32        CanvasLeft = 10;    // left of the display area
static Int32        AcqWinWidth;
static Int32        AcqWinHeight;
static BOOL                         StopGrab = FALSE;
static unsigned int plotFlag;


int N = 1024;
//Int16 acq_image[1024][512];
uInt16 output_image[1024][512];//output_image[512][256];
double index[1024];
double default_index[1024];
Int16 interpolate_index[1024];
double lambda[1024];
double interpolate_array[8][1024];

int a, b;
int Na = 8, Nb = 1024;
int k2 = 0;
double k0[1024], k1[1024], k1_inv[1024], kmax = 0, kmin = 0;
double line_rate = 91912;
int Aline_Num = 1000;
double amp = 1;   //2.5
double Bperiod = 0;//1000 * Aline_Num * 2 / line_rate; //In microseconds;
double Bfreq = 0;

int peakDepth = 0;

DWORD FRIMAINID;
static HANDLE FRIMAINThread;
DWORD GIVEID;
static HANDLE giveThread;
double *InterpolateVector;
fftw_plan fftPlan;
fftw_complex *in, *out;
uInt16 *fftOutput;
uInt16 *fftOutputFinal;
int allow;
//THINGS FOR THE DISPLAY END-------------------------------------------------------------------

using namespace KUKA::FRI;

//Galvo variables
#define DEFAULT_PORTID 30200
#define DEFAULT_JOINTMASK 0x20
#define DEFAULT_FREQUENCY 0.1
#define DEFAULT_AMPLITUDE 0.01
#define DEFAULT_FILTER_COEFFICIENT 0.7
#define PI       3.1415926535
#define Cspeed = 299792458;
#define DAQmxErrChk(functionCall) if( DAQmxFailed(error=(functionCall)) ) goto Error; else
#define errChk(fCall) if (error = (fCall), error < 0) {goto Error;} else
int32 CVICALLBACK DoneCallback(TaskHandle taskHandle, int32 status, void *callbackData);
//End of galvo variables

//CODE FOR THE DISPLAY, THIS CONTAINS THE MAIN-----------------------------------------
------------------------------
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
        LPSTR lpszCmdLine, int nCmdShow)
{
        CHAR       ImaqSmplClassName[] = "Imaq Sample";
        WNDCLASS             ImaqSmplClass;
        MSG                  msg;
        HWND       hTemp;

        // register the main window
```

```
	hInst = hInstance;
	if (!hPrevInstance)
	{
		ImaqSmplClass.style = CS_HREDRAW | CS_VREDRAW;
		ImaqSmplClass.lpfnWndProc = (WNDPROC)ImaqSmplProc;
		ImaqSmplClass.cbClsExtra = 0;
		ImaqSmplClass.cbWndExtra = 0;
		ImaqSmplClass.hInstance = hInstance;
		ImaqSmplClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
		ImaqSmplClass.hCursor = LoadCursor(NULL, IDC_ARROW);
		ImaqSmplClass.hbrBackground                                        =
static_cast<HBRUSH>(GetStockObject(LTGRAY_BRUSH));
		ImaqSmplClass.lpszMenuName = 0;
		ImaqSmplClass.lpszClassName = ImaqSmplClassName;

		if (!RegisterClass(&ImaqSmplClass))
			return (0);
	}

	// creates the main window
	ImaqSmplHwnd        =        CreateWindow(ImaqSmplClassName,        "HLGrab",
WS_OVERLAPPEDWINDOW | WS_VISIBLE,
		CW_USEDEFAULT, CW_USEDEFAULT, 700, 600, NULL, NULL, hInstance,
NULL);


	// creates the interface name label
	if (!(hTemp = CreateWindow("Static", "Interface Name", ES_LEFT | WS_CHILD |
WS_VISIBLE,
		540, 14, 100, 20, ImaqSmplHwnd, (HMENU)-1, hInstance, NULL)))
		return(FALSE);

	// creates the frame rate label
	if (!(hTemp = CreateWindow("Static", "Frame Rate", ES_LEFT | WS_CHILD |
WS_VISIBLE,
		540, 232, 140, 20, ImaqSmplHwnd, (HMENU)-1, hInstance, NULL)))
		return(FALSE);

	if (!(CreateWindow("Static", "Peak Depth", ES_LEFT | WS_CHILD | WS_VISIBLE,
		540, 280, 140, 20, ImaqSmplHwnd, (HMENU)-1, hInstance, NULL)))
		return(FALSE);

	// creates the interface name edit box
	if  (!(HIntfName  =  CreateWindow("Edit",  "img0",  ES_LEFT  |  WS_CHILD  |
WS_VISIBLE | WS_BORDER,
		540, 34, 100, 20, ImaqSmplHwnd, (HMENU)-1, hInstance, NULL)))
		return(FALSE);
	// creates the frame rate edit box
	if (!(HFrameRate = CreateWindow("Edit", "0", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER,
		540, 252, 100, 20, ImaqSmplHwnd, (HMENU)-1, hInstance, NULL)))
		return(FALSE);

	if (!(HPeakPosition = CreateWindow("Edit", "0", ES_LEFT | WS_CHILD | WS_VISIBLE
| WS_BORDER,
		540, 300, 100, 20, ImaqSmplHwnd, (HMENU)-1, hInstance, NULL)))
		return(FALSE);

	// creates the Grab button
	if (!(HGrab = CreateWindow("Button", "Grab", BS_PUSHBUTTON | WS_CHILD |
WS_VISIBLE | WS_BORDER,
		540, 72, 80, 40, ImaqSmplHwnd, (HMENU)PB_GRAB, hInstance, NULL)))
		return(FALSE);

	// creates the stop button
	if (!(HStop = CreateWindow("Button", "Stop", BS_PUSHBUTTON | WS_CHILD |
WS_VISIBLE | WS_BORDER,
		540, 112, 80, 40, ImaqSmplHwnd, (HMENU)PB_STOP, hInstance, NULL)))
		return(FALSE);

	EnableWindow(HStop, FALSE);
	EnableWindow(HFrameRate, FALSE);

	// creates the quit application button
	if (!(HQuit = CreateWindow("Button", "Quit", BS_DEFPUSHBUTTON | WS_CHILD |
WS_VISIBLE,
		540, 152, 80, 40, ImaqSmplHwnd, (HMENU)PB_QUIT, hInstance, NULL)))
		return(FALSE);

	//Initialize a vital array
	for (int a = 0; a < Na; a++) {
		for (int b = 0; b < Nb; b++) {
			interpolate_array[a][b] = (double)1.0;
		}
	}

	// Display the main window
	ShowWindow(ImaqSmplHwnd, SW_SHOW);
	UpdateWindow(ImaqSmplHwnd);


	while (GetMessage(&msg, NULL, 0, 0))
	{
		TranslateMessage(&msg);
		DispatchMessage(&msg);
```

```
        }

        // Wait for the stop thread to complete before returning
        WaitForSingleObject(HStopThread, INFINITE);

        return (int)(msg.wParam);
}


// Message proc
LRESULT CALLBACK ImaqSmplProc(HWND hWnd, UINT iMessage, UINT wParam, LONG
lParam)
{
        WORD            wmId;

        switch (iMessage)
        {
        case WM_COMMAND:
                wmId = LOWORD(wParam);
                switch (wmId)
                {
                case PB_QUIT:
                        PostQuitMessage(0);
                        break;
                case PB_GRAB:
                        // Grab button has been pressed
                        OnGrab();
                        break;
                case PB_STOP:
                        // Grab button has been pressed
                        SetEvent(HStopEvent);
                        break;
                }
                break;
        case WM_DESTROY:
                SetEvent(HStopEvent);
                PostQuitMessage(0);

        default:
                return DefWindowProc(hWnd, iMessage, wParam, lParam);
                break;
        }
        return 0;
}


// Function executed when the grab button is clicked
int OnGrab(void)
```

```
{
	int		error;
	char		intfName[64];
	unsigned int	bitsPerPixel;
	DWORD				dwThreadId;

	struct mainType FRIMAINStruct;
	FRIMAINStruct.argc = 1;
	FRIMAINStruct.argv;

	FRIMAINThread = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)FRIMAIN, &FRIMAINStruct, 0, &FRIMAINID);
	if (!FRIMAINThread)
		return 0;

	// Create the event that needs to be signaled when we
	// wish to stop the acquisition.
	HStopEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
	if (!HStopEvent)
		return 0;

	// Create the thread that is responsible for shutting
	// down the acquisition
	HStopThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)StopThread,
(LPDWORD)&HStopEvent, 0, &dwThreadId);
	if (!HStopThread)
		return 0;

	// Get the interface name
	GetWindowText(HIntfName, intfName, 64);

	// Open an interface and a session
	errChk(imgInterfaceOpen(intfName, &Iid));
	errChk(imgSessionOpen(Iid, &Sid));

	// Let's check that the Acquisition window is not smaller than the Canvas
	errChk(imgGetAttribute(Sid, IMG_ATTR_ROI_WIDTH, &AcqWinWidth));
	errChk(imgGetAttribute(Sid, IMG_ATTR_ROI_HEIGHT, &AcqWinHeight));

	if (CanvasWidth < AcqWinWidth)
		AcqWinWidth = CanvasWidth;
	if (CanvasHeight < AcqWinHeight)
		AcqWinHeight = CanvasHeight;

	// get the pixel depth of the camera.
	errChk(imgGetAttribute(Sid, IMG_ATTR_BITSPERPIXEL, &bitsPerPixel));

	switch (bitsPerPixel)
	{
	case 10:
		plotFlag = IMGPLOT_MONO_10;
		break;
	case 12:
		plotFlag = IMGPLOT_MONO_12;
		break;
	case 14:
		plotFlag = IMGPLOT_MONO_14;
		break;
	case 16:
		plotFlag = IMGPLOT_MONO_16;
		break;
	case 24:
	case 32:
		// assumes that a 24 bits camera is a color camera.
		// in this mode, even if the camera is 24 bits the board returns 32 bits values
		plotFlag = IMGPLOT_COLOR_RGB32;
		break;
	default:
		plotFlag = IMGPLOT_MONO_8;
		break;
	}

	// Set the ROI to the size of the Canvas so that it will fit nicely
	errChk(imgSetAttribute2(Sid, IMG_ATTR_ROI_WIDTH, N));
	errChk(imgSetAttribute2(Sid, IMG_ATTR_ROI_HEIGHT, AcqWinHeight));
	errChk(imgSetAttribute2(Sid, IMG_ATTR_ROWPIXELS, N));


	// Setup and launch the grap operation
	errChk(imgGrabSetup(Sid, TRUE));

	// Let NI-IMAQ manage the memory
	ImaqBuffer = NULL;
	Bperiod = 1000 * Aline_Num * 2 / line_rate; //In microseconds
	Bfreq = line_rate / (Aline_Num * 4);

	StopGrab = FALSE;


	// Start the acquisition thread
	HThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ImaqThread,
(LPDWORD*)&StopGrab, 0, &dwThreadId);
	if (HThread == NULL)
		return 0;

	EnableWindow(HStop, TRUE);
```

```
        EnableWindow(HGrab, FALSE);
        EnableWindow(HQuit, FALSE);
Error:
        if (error<0) {
                DisplayIMAQError(error);
                PostMessage(ImaqSmplHwnd, WM_COMMAND, PB_STOP, 0);
        }

        return 0;
}


DWORD ImaqThread(LPDWORD lpdwParam)
{
        static int nbFrame = 0, error;
        static int t1, t2;
        char            buffer[32];
        //char          buffer2[32];
        int Aline_count = 512;
        int Pixel_count = N;
        int peakDepthNum = 0;
        int avg[1024];
        char temp[10];
        double startValue = (2 * PI) / (1189 * 0.00000001);
        double *la;
        double k1, k2;
        double step;
        double magnitude;

        peakDepth = 0;

        BOOL line_order = FALSE;
        // Create a pointer to the stop boolean. This needs to be
        // volatile because the value can change at any time.
        BOOL* volatile stop = (BOOL*)lpdwParam;

        //errChk(imgGrabSetup(Sid, TRUE));

        in = (fftw_complex *)malloc(sizeof(fftw_complex) * N);
        out = (fftw_complex *)malloc(sizeof(fftw_complex) * N);
        fftOutput = (uInt16 *)malloc(sizeof(uInt16) * (N / 2) * Aline_count);
        fftOutputFinal = (uInt16 *)malloc(sizeof(uInt16) * (N / 2) * Aline_count);
        InterpolateVector = (double *)malloc(sizeof(double) * N);
        la = (double *)malloc(sizeof(double) * N);

        fftPlan = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_MEASURE);

        for (int a = 0; a < N; a++) {
```

```
		la[a] = 0.000000001 * (1188.89 + (0.2268 * a) - (0.0000082 * pow(a, 2) -
(0.00000000699 * pow(a, 3)))); //the wavelength x axis.
	}

	k1 = la[0]; //min
	k2 = la[0]; //max

	for (int a = 0; a < N; a++) {
		if (k1 > la[a]) {
			k1 = la[a];
		}
		if (k2 < la[a]) {
			k2 = la[a];
		}
	}

	k1 = (2 * PI) / k1;
	k2 = (2 * PI) / k2;

	step = (abs(k1 - k2)) / 1024;

	for (int a = 0; a < 1024; a++) {
		avg[a] = 0;
	}

	// the thread stop when StopRing goes to TRUE or there is an error
	while (!*stop && !error) {
		allow = 0;
		t2 = GetTickCount();

		// Get the frame after tne next Vertical Blank
		errChk(imgGrab(Sid, (void **)&ImaqBuffer, TRUE));

		for (int x = 0; x < N; x++) {
			for (int y = 0; y < Aline_count; y++) {
				avg[x] = avg[x] + (uInt16)*(ImaqBuffer + y*N + x);
			}

			avg[x] = avg[x] / Aline_count;
		}

		for (int y = 0; y < Aline_count; y++) {
			for (int x = 0; x < N; x++) {
				in[x][0] = (uInt16)*(ImaqBuffer + y*N + x) - avg[x];
				in[x][1] = 0.0; //no imaginary part.
			}

			//for (int x = 0; x < N; x++) {
					//(double)*(InterpolateVector + x) = in[x][0];
				//}

				//boost::math::cubic_b_spline<double>	spline(InterpolateVector,	N,
startValue, step);

				//for (int x = 0; x < N; x++) {
					//if (spline((2 * PI) / la[x]) < 0) {
						//in[x][0] = 0;
					//}
					//else {
						//in[x][0] = spline((2 * PI) / la[x]);
					//}
				//}

				fftw_execute(fftPlan);

				for (int x = 0; x < N / 2; x++) {
					magnitude = sqrt(out[x][0] * out[x][0] + out[x][1] * out[x][1]);

					(uInt16)*(fftOutput + y*(N / 2) + x) = (uInt16)magnitude;
				}
			}

			peakDepthNum = 0;
			peakDepth = 0;

			for (int y = 0; y < Aline_count; y++) {
				for (int x = 50; x < N / 2; x++) {
					if (peakDepthNum < (uInt16)*(fftOutput + y*(N / 2) + x)) {
						peakDepthNum = (uInt16)*(fftOutput + y*(N / 2) + x);
						peakDepth = x;
					}
				}
			}

			//peakDepth = (peakDepth / 5) * 5;

			for (int y = 0; y < Aline_count; y++) {
				for (int x = 0; x < N / 2; x++) {
					(uInt16)*(fftOutputFinal + y*(N / 2) + x) = (uInt16)*(fftOutput +
x*(N / 2) + y);
				}
			}

			// Display it using imgPlot
			// Note that if you are using a board or camera with bitdepth greater
			// that 8 bits, you need to set the flag parameter of imgPlot to match
```

```
                // the bit depth of the camera. See the "snap imgPlot" sample.
                //int data = (uInt16)*(ImaqBuffer + 500 * N + 450);

                imgPlot2(ImaqSmplHwnd, fftOutputFinal, 0, 0, AcqWinWidth, AcqWinHeight,
                        CanvasLeft, CanvasTop, plotFlag);

                allow = 1;

                itoa(peakDepth, temp, 10);

                //set the value of the peak position on the display window.
                SetWindowText(HPeakPosition, temp);

                // Calculate the number of frame per seconds every 10 frames
                nbFrame++;
                if (nbFrame>10)
                {
                        sprintf(buffer, "%.2f", 1000.0 * (double)nbFrame / (double)(t2 - t1));
                        SetWindowText(HFrameRate, buffer);
                        t1 = t2;
                        nbFrame = 0;
                }

        Error:
                if (error<0 && !*stop)
                {
                        DisplayIMAQError(error);
                        PostMessage(ImaqSmplHwnd, WM_COMMAND, PB_STOP, 0);
                }

        }
        free(la);
        return 0;
}


// Waits for the stop event to occur, then stops the acquisition.
DWORD StopThread(LPDWORD lpdwParam) {
        DWORD dwResult;

        // Get a handle to the stop event
        HANDLE event = *((HANDLE*)lpdwParam);

        // Wait for the done event to occur
        dwResult = WaitForSingleObject(event, INFINITE);
        if (dwResult != WAIT_FAILED) {
                CloseHandle(event);
                event = NULL;
```

```
	}

	// Stop the thread
	StopGrab = TRUE;

	// Wait for the thread to end and kill it otherwise
	dwResult = WaitForSingleObject(HThread, 2000);
	if (dwResult == WAIT_TIMEOUT)
		TerminateThread(HThread, 0);

	// Stop the acquisition
	imgSessionStopAcquisition(Sid);

	// Close the interface and the session
	if (Sid != 0)
		imgClose(Sid, TRUE);
	if (Iid != 0)
		imgClose(Iid, TRUE);

	EnableWindow(HStop, FALSE);
	EnableWindow(HGrab, TRUE);
	EnableWindow(HQuit, TRUE);

	free(InterpolateVector);
	free(in);
	free(out);
	free(fftOutput);
	free(fftOutputFinal);
	fftw_destroy_plan(fftPlan);

	return 0;
}



// in case of error this function will display a dialog box
// with the error message
void DisplayIMAQError(Int32 error)
{
	static Int8 ErrorMessage[256];

	memset(ErrorMessage, 0x00, sizeof(ErrorMessage));

	// converts error code to a message
	imgShowError(error, ErrorMessage);

	MessageBox(ImaqSmplHwnd, ErrorMessage, "Imaq Sample", MB_OK);
}
```

```
//CODE FOR THE DISPLAY, THIS CONTAINS THE MAIN----------------------------------------
-----------------------------


int FRIMAIN(struct mainType *FRIMAINStruct)
{
	//----------------------------------------------------------------------------------------------------
-----
	int argc = FRIMAINStruct->argc;
	char **argv = FRIMAINStruct->argv;

	/*LPDWORD lpdwParam = 0;
	TaskHandle	taskHandle11 = (void*)1;   //Task 1	set both channel to 0
	float64	datainitiate[2] = { 0 };
	DAQmxCreateTask("", &taskHandle11);
	DAQmxCreateAOVoltageChan(taskHandle11, "Dev2/ao0:Dev2/ao1", "", -10.0, 10.0,
DAQmx_Val_Volts, "");
	DAQmxStartTask(taskHandle11);
	DAQmxWriteAnalogF64(taskHandle11, 1, 1, 10.0, DAQmx_Val_GroupByChannel,
datainitiate, NULL, NULL);
	DAQmxStopTask(taskHandle11);
	DAQmxClearTask(taskHandle11);
	//----------------------------------------------------------------------------------------------------
------
	static int nbFrame = 0, nbFrame1 = 0;
	static int t1, t2;

	char		buffer[32];
	char		buffer2[32];
	int Aline_count = 0;
	int Pixel_count = 0;
	//float64 data2[16000];

	BOOL line_order = FALSE;

	TaskHandle  taskHandle2;
	double data1 = 0;
	DAQmxCreateTask("", &taskHandle2);
	DAQmxCreateCOPulseChanFreq(taskHandle2, "/Dev1/ao0", "", DAQmx_Val_Hz,
DAQmx_Val_Low, data1, Bfreq, 0.5);
	DAQmxCfgImplicitTiming(taskHandle2, DAQmx_Val_ContSamps, 1000);
	DAQmxRegisterDoneEvent(taskHandle2, 0, DoneCallback, NULL);
	DAQmxStartTask(taskHandle2);

	//NIDAQmx declarations
	int32	error = 0;
	TaskHandle  taskHandle = 0;
	//	TaskHandle  taskHandle1 = 3;
	int samnum = 1000;
	float64	data[16000]; //12290 = real_period * 12, originally 2000 for normal wave with
one galvo. 1 triangle = 2000
							//uInt8		data1[212+2];
	char	errBuff[2050] = { '\0' };
	int	i = 0;
	int			k = 0;
	int			kand = 0; //Master counter for number of triangles
	int			j = 1; //for toggle 1 to -1 for creating triangle waveform

	double		slop = 2 * amp / samnum;
	double		slop2 = (0.666666*amp) / samnum; //slope for increasing segments in
secondary galvo driver signal
	double		oneOverThree = 0.333333;
	BOOL* volatile stop = (BOOL*)lpdwParam;

	//int32	error = 0;
	TaskHandle  taskHandle1 = (void*)3;

	//Dual wave for two galvos "c-scan" mode, must initiate both channels "/Dev1/ao0" ->
"/Dev1/ao0:1"
	//"kand < 8" where 8/2 is the number of triangles, so "kand < 4" will make two triangles
	for (kand = 0; kand < 8; kand++) {
		for (i = 0; i < samnum; i++) {
			data[i + kand * 1000] = -j*amp + j*i*slop;
		}
		j = -1 * j;
	}

	//Waveform for secondary galvo for four triangles will need strong modification if number
of triangles is changed
	for (i = 0; i < samnum; i++)
		data[i + 8000] = -amp; //hold position while acquire
	for (i = 0; i < samnum; i++)
		data[i + 9000] = i*slop2 - amp; //segment 1
	for (i = 0; i < samnum; i++)
		data[i + 10000] = -oneOverThree*amp;
	for (i = 0; i < samnum; i++)
		data[i + 11000] = i*slop2 - oneOverThree*amp; //segment 2
	for (i = 0; i < samnum; i++)
		data[i + 12000] = oneOverThree*amp;
	for (i = 0; i < samnum; i++)
		data[i + 13000] = i*slop2 + oneOverThree*amp; //segment 3
	for (i = 0; i < samnum; i++)
		data[i + 14000] = amp;
	for (i = 0; i < samnum; i++)
		data[i + 15000] = (-2 * amp / samnum)*i + amp; //return segment
```

```
/*********************************************/

// DAQmx Configure Code

/*********************************************/

	/*DAQmxCreateTask("", &taskHandle);
	DAQmxCreateAOVoltageChan(taskHandle,    "/Dev2/ao0:1",    "",    -10.0,    10.0,
DAQmx_Val_Volts, NULL);
	DAQmxCfgSampClkTiming(taskHandle,    "",    line_rate,    DAQmx_Val_Rising,
DAQmx_Val_ContSamps, samnum * 2);
	DAQmxRegisterDoneEvent(taskHandle, 0, DoneCallback, NULL);


	/*********************************************/
	// DAQmx Configure Code
	/*********************************************/
	/*DAQmxCreateTask("", &taskHandle1);
	DAQmxCreateCOPulseChanFreq(taskHandle1,    "Dev2/Ctr0",    "",    DAQmx_Val_Hz,
DAQmx_Val_Low, 0, Bfreq * 2, 0.5);
	DAQmxCfgImplicitTiming(taskHandle1, DAQmx_Val_ContSamps, 1000);

	DAQmxRegisterDoneEvent(taskHandle1, 0, DoneCallback, NULL);
	/*********************************************/
	// DAQmx Write Code
	/*********************************************/
	//DAQmxWriteAnalogF64(taskHandle,    samnum    *    8,    0,    10.0,
DAQmx_Val_GroupByChannel, data, NULL, NULL);
	/*********************************************/
	// DAQmx Write Code
	/*********************************************/
	//DAQmxStartTask(taskHandle);

	/*********************************************/
	// DAQmx Start Code
	/*********************************************/

	//DAQmxStartTask(taskHandle1);

	//------------------------------------------------------------------------------------------------------
-----

	// parse command line arguments
	if (argc > 1)
	{
		if (strstr(argv[1], "help") != NULL)
		{
			printf(
				"\nKUKA LBR joint sine overlay test application\n\n"
				"\tCommand line arguments:\n"
				"\t1) remote hostname (optional)\n"
				"\t2) port ID (optional)\n"
				"\t3) bitmask encoding of joints to be overlayed (optional)\n"
				"\t4) sine frequency in Hertz (optional)\n"
				"\t5) sine amplitude in radians (optional)\n"
				"\t6) filter coefficient from 0 (off) to 1 (optional)\n"
			);
			return 1;
		}
	}
	char* hostname = (argc >= 2) ? argv[1] : NULL;
	int port = (argc >= 3) ? atoi(argv[2]) : DEFAULT_PORTID;
	unsigned int jointMask = (argc >= 4) ? (unsigned int)atoi(argv[3]) :
DEFAULT_JOINTMASK;
	double frequency = (argc >= 5) ? atof(argv[4]) : DEFAULT_FREQUENCY;
	double amplitude = (argc >= 6) ? atof(argv[5]) : DEFAULT_AMPLITUDE;
	double filterCoeff = (argc >= 7) ? atof(argv[6]) : DEFAULT_FILTER_COEFFICIENT;

	/*********************************************************************
****/
	/*                                                  */
	/*   Place user Client Code here                              */
	/*                                                  */
	/*********************************************************************
***/
	amplitude = DEFAULT_AMPLITUDE + 0.1;
	// create new sine overlay client
	LBRJointSineOverlayClient client(jointMask, frequency, amplitude, filterCoeff);

	giveThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)sendToClient,
&client, 0, &GIVEID);
	if (!giveThread)
		return 0;

	/*********************************************************************
****/
	/*                                                  */
	/*   Standard application structure                           */
	/*   Configuration                                       */
	/*                                                  */
	/*********************************************************************
****/

	// create new udp connection
	UdpConnection connection;


	// pass connection and client to a new FRI client application
	ClientApplication app(connection, client);

	// connect client application to KUKA Sunrise controller
	app.connect(port, hostname);

	/*************************************************************************
****/
	/*                                              */
	/*   Standard application structure                         */
	/*   Execution mainloop                                  */
	/*                                              */
	/*************************************************************************
****/

	// repeatedly call the step routine to receive and process FRI packets
	bool success = true;
	while (success)
	{
		success = app.step();
	}

	//DAQmxStopTask(taskHandle);
	//DAQmxClearTask(taskHandle);
	//DAQmxStopTask(taskHandle2);
	//DAQmxClearTask(taskHandle2);


	/*************************************************************************
****/
	/*                                              */
	/*   Standard application structure                         */
	/*   Dispose                                        */
	/*                                              */
	/*************************************************************************
****/

	// disconnect from controller
	app.disconnect();

	return 1;
}

void sendToClient(LBRJointSineOverlayClient *function) {
	while (true) {
		//if (allow == 1) {
			function->giveDepth(peakDepth);
		//}
	}
}

int32 CVICALLBACK DoneCallback(TaskHandle taskHandle, int32 status, void *callbackData)
{
	int32   error = 0;
	char    errBuff[2048] = { '\0' };

	// Check to see if an error stopped the task.
	DAQmxErrChk(status);

Error:
	if (DAQmxFailed(error)) {
		DAQmxGetExtendedErrorInfo(errBuff, 2048);
		DAQmxClearTask(taskHandle);
		printf("DAQmx Error: %s\n", errBuff);
	}
	return 0;
}
```

/*************************** LBROverlayClient.cpp *****************************/***

```
#include <cstring>
#include <cstdio>
#include "LBRJointSineOverlayClient.h"
#include "friLBRState.h"
#include <iostream>
#include <fstream>
#include <math.h>
#include "pid.h"


// Visual studio needs extra define to use math constants
#define _USE_MATH_DEFINES
#include <cmath>

int newPeakDepth;

std::ofstream errorFile;
std::ofstream phi;
```

```
using namespace KUKA::FRI;

PID pidController = PID(0.001, 0.1, 0, 0.020, 0,0);

//---------------------------------------------------------------------------
void LBRJointSineOverlayClient::giveDepth(int depth) { //using this function, we can transfer any
variable that we would like.
        newPeakDepth = depth;
}
//---------------------------------------------------------------------------

//******************************************************************************
**
LBRJointSineOverlayClient::LBRJointSineOverlayClient(unsigned int jointMask,
    double freqHz, double amplRad, double filterCoeff)
  : _jointMask(jointMask)
  , _freqHz(freqHz)
  , _amplRad(amplRad)
  , _filterCoeff(filterCoeff)
  , _offset(0.0)
  , _phi(0.0)
  , _stepWidth(0.0)
{
  /*printf("LBRJointSineOverlayClient initialized:\n"
      "\tjoint mask: 0x%x\n"
      "\tfrequency (Hz): %f\n"
      "\tamplitude (rad): %f\n"
      "\tfilterCoeff: %f\n",
      jointMask, freqHz, amplRad, filterCoeff);*/

        //amprad is peakDepth;
}

//******************************************************************************
**
LBRJointSineOverlayClient::~LBRJointSineOverlayClient()
{
}

//******************************************************************************
**
void LBRJointSineOverlayClient::onStateChange(ESessionState oldState, ESessionState
newState)
{
  LBRClient::onStateChange(oldState, newState);
  // (re)initialize sine parameters when entering Monitoring
  switch (newState)
  {
    case MONITORING_READY:
    {
      _offset = 0.0;
      _phi = 0.0;
      _stepWidth =  robotState().getSampleTime(); //2 * M_PI * _freqHz *
                  errorFile.open("error.txt");
                  phi.open("phi.txt");
      break;
    }
    default:
    {
      break;
    }
  }
}



//*****************************************************************************
**
void LBRJointSineOverlayClient::command()
{

        double error;
        double step;

        //error = ((((double)newPeakDepth - 100) / 666) * 3.149) / 180 ;
        error = ((double)newPeakDepth - 250) * 0.000004;

        // calculate new offset
        //double newOffset = _amplRad *sin(_phi);
        //double newOffset = _phi;
        //double newOffset = _phi;
        //_offset = _offset * _filterCoeff + newOffset * (1.0 - _filterCoeff);
        //step = pidController.calculate(0, error);
        /*step = 0.005*_stepWidth;
        if (error < 0) {

                if (error > -0.0005) {
                        step = 0.00000001*_stepWidth;
                }
                _phi -= step;

        }
        else if (error > 0){
                if (error < 0.0005) {
                        step = 0.00000001*_stepWidth;
                }
```

```
                        _phi += step;
        }*/

        step = pidController.calculate(0, -1*abs(error));
        //_phi += step;

        if (error < 0) {
                _phi -= step;

        }
        else if (error > 0) {
                _phi += step;
        }

        //step = sqrt((0.0002)-((1/4)*abs(error)-0.0142)*((1/4)*abs(error) - 0.0142))*_stepWidth;
        /*step = abs(error) * _stepWidth;

        if (error < 0) {
                _phi -= step;

        }
        else if (error > 0) {
                _phi += step;
        }*/

        errorFile << error << "\n";
        phi << step << "\n";


        //_phi += 0.1*_stepWidth;

        //if (_phi >= (1/16) * M_PI) _phi -= (1/16) * M_PI;
        //if (_phi >= error) _phi = 0;

        // add offset to ipo joint position for all masked joints
        double jointPos[LBRState::NUMBER_OF_JOINTS];
        memcpy(jointPos, robotState().getIpoJointPosition(), LBRState::NUMBER_OF_JOINTS
* sizeof(double));
        for (int i=0; i< LBRState::NUMBER_OF_JOINTS; i++)
        {
          if (_jointMask & (1<<i))
          {

                        jointPos[i] += _phi;
                //printf("JointPos[6]: %f\n", jointPos[i]);
        }
      }
      //double jointPos[LBRState::NUMBER_OF_JOINTS] = {0, 0, 0, 0, 0, 0.6, 0};

  robotCommand().setJointPosition(jointPos);

}
//***************************************************************************
**
// clean up additional defines
#ifdef _USE_MATH_DEFINES
#undef _USE_MATH_DEFINES
#endif
```

Reference list

[1] M. Lara-Velazquez, A. Quinones-Hinojosa, “Advances in brain tumor surgery of glioblastoma in adults”, Brain Sciences, 2017

[2] S.A. Boppart, “Optical coherence tomography: technology and applications for neuroimaging”

[3] Bohringer, “Non-invasive intraoperative optical coherence tomography of the resection cavity during surgery of intrinsic brain tumors”, Proceedings of the Photonics Therapeutics and Diagnosis, 2006

[4] Bohringer “Imaging of human brain tumor tissue by near-infrared laser coherence tomography”, 2009

[5] C. Kut, A. Quiñones-Hinojosa, “Detection of human brain cancer infiltration ex vivo and in vivo using quantitative optical coherence tomography,” Sci. Transl. Med. 7(292), 292ra100 (2015).

[6] R.M. Juarez-Chambi, A. Quiñones-Hinojosa, “Detection of brain tumor margins using optical coherence tomography”, SPIE Medical Imaging 2018

[7] J.G. Fujimoto, “Optical biopsy and imaging using optical coherence tomography”, Nature Med, 1995

[8] A.F. Fercher, W. Drexler, C.K. Hitzenberger, “Optical coherence tomography – principles and applications”,

[9] J.A. Izatt, “Optical Coherence Tomography for biodiagnostics”, 1997

[10] G.L. Monroy, S.A. Boppart, “Clinical translation of handheld optical coherence tomography: practical considerations and recent advancements”, Journal of Biomedical Optics, Dec 2017

[11] Cold-Ablation Robot-guided Laser Osteotome (CARLO) aot.swiss/en/carlo/ visited on November 2019

[12] Accuray https://www.accuray.com/ visited on November 2019

[13] Siemens Healthineers https://www.siemens-healthineers.com visited on November 2019

[14] M. E. Wagshul, P. K. Eide, J. R. Maiden, “The pulsating brain: A review of experimental and clinical studies of intracranial pulsatility”, Fluids Barriers CNS, 2011; 8: 5

[15] M. Bianciardi, N. Toschi, J. R. Polimeni, K.C. Evans, H. Bhat, B. Keil, B. R. Rosen, D. A. Boas, “The pulsatility volume index: an indicator of cerebrovascular compliance based on fast magnetic resonance imaging of cardiac and respiratory pulsatility”, Philo Trans A Math Phys Eng Sci, 2016 May 13; 374(2067): 20150184.

[16] J.M. Dudley, J.R. Taylor, “Ten years of nonlinear optics in photonic crystal fibers”, Nature Photonics 2009

[17] Y. Lim, T. Yatagai, Y. Otani, “Ultrahigh resolution spectral domain optical coherence tomography using supercontinuum light source”, Opt Rev 2016

[18] Y. Wang, Y. Zhao, J.S. Nelson, Z, Chen, “Ultrahigh-resolution optical coherence tomography by broadband continuum generation from a photonic crystal fiber”, Optics Letters, Vol 28, No 3, February 2003

[19] Y. Nomura, H. Kawagoe, N. Nishizawa, “Supercontinuum generation for ultrahigh-resolution optical coherence tomography at wavelength of 0.8μm using carbon nanotube fiber laser and similariton amplifier”, Applied Physics Express, 2014

[20] Y.J. You, C. Wang, Y.L. Lin, A. Zaytsev, P. Xue, C.L. Pan, “Ultrahigh-resolution optical coherence tomography 1.3μm central wavelength y using a supercontinuum source pumped y noise-like pulses”, Laser Physics Letters, 2016

[21] M.L. Ferhat, L. Cherbi, “Supercontinuum generation inn optimized photonic crystal fiber at 1.3μm for optical coherence tomography”, CSNDD 2016

[22] M. Yamanaka, H. Kawagoe, N. Nishizawa, “High-power supercontinuum generation using high-repetition-rate ultrashort-pulse fiber laser for ultrahigh-resolution optical coherence tomography in 1600nm spectral band”, Applied Physics Express, Vol 9, 2016

[23] H. Kawagoe, S. Ishida, M. Aramaki, Y. Sakakibara, E. Omoda, H. Kataura, N. Nishizawa, “Development of a high-power supercontinuum source in the 1.7μm wavelength region for highly penetrative ultrahigh-resolution optical coherence tomography”, Biomedical Optical Express, Vol 5, No 3, February 2014

[24] H. Kawagoe, N. Nishizawa, “Ultrahigh-resolution optical coherence tomography using supercontinuum source in 1.9μm wavelength region”, Optical Society of America, 2014

[25] S. O’Sullivan, H. Ashrafian, “Legal, regulatory, and ethical frameworks for development of standards in artificial intelligence and autonomous robotic surgery”, International Journal of Medical Robotics and Computer Assisted Surgery, 2018

[26] SAFROS project (Patient safety in robotic surgery), FP7-ICT grant agreement No. 248960. www.safros.eu visited on Nov 2019

[27] E.PalmeriniaA, BertoliniaF, BattagliabB.-J, KoopscA. CarnevaleaP. Salvinia “RoboLaw: Towards a European framework for robotics regulation”, Robotics and Autonomous Systems Volume 86, December 2016, Pages 78-85

[28] Regulating Emerging Robotic Technologies in Europe: Robotics facing Law and Ethics www.robolaw.eu visited on November 2019

[29] J. Guiochet, M. Machin, H. Waeselynck, “Safety-critical advanced robots: a survey”, Robotics and Autonomous Systems, 2017

[30] A. Mylaeus, A. Vempati, “A survey on the pain threshold and its use in robotics safety standards”

[31] M. Y. Park, D. Han, J. H. Lim, M. K. Shin, Y. R. Han, D. H. Kim, S. Rhim, K. S. Kim, “Assessment of pressure pain thresholds in collisions with collaborative robots”, PLOS ONE, May 2019

[32] J. Falco, J. Marvel, R. Norcross, “Collaborative Robotics: Measuring Blunt Force Impacts on Humans”, National Institute of Standards and Technology

[33] M. Finke, Scheiweikard, “Automatic scanning of large tissue areas in neurosurgery using optical coherence tomography”, International Journal of Medical Robotics and Computer Assisted Surgery, 2012

[34] S.V. Kantelhardt, M. Finke, A. Schweikard, A. Giese, “Evaluation of a completely robotized neurosurgical operating microscope”, Neurosurgery, Vol 72, Jan 2013

[35] M. Draelos, P. Ortiz, R. Qian, B. Keller, K. Hauser, A. Kuo, . Izatt, “Automatic Optical Coherence Tomography Imaging of Stationary and Moving Eyes with a robotically-aligned scanner”,
[36] L. Sakka, G. Coll b, J. Chazal, “Anatomy and physiology of cerebrospinal fluid”, European Annals of Otorhinolaryngology, Head and Neck diseases (2011) 128, 309—316cs

[37] M Czosnyka, J D Pickard, “Monitoring and interpretation of intracranial pressure”, J Neurol Neurosurg Psychiatry 2004;75:813–821.

[38] F. Wadehn, D.J. Mack, E. Keller, T. Heldt, “Multiscale Intracranial Pressure Signal Simulator”, Computing in Cardiology Conference, Sept 2018

[39] M. Ursino, C.A. Lodi, “A simple mathematical model of the interaction between intracranial pressure and cerebral hemodynamics”,