Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Nikolai Stensø

Characterization and Signal Processing of Mach-Zehnder Assisted Ring Resonator Configuration Sensors

Master's thesis in Nanotechnology
Supervisor: Astrid Aksnes
Co-supervisor: Arnfinn Aas Eielsen, Jens Høvik
June 2022

NTNU
Norwegian University of Science and Technology

Nikolai Stensø

Characterization and Signal Processing of Mach-Zehnder Assisted Ring Resonator Configuration Sensors

Master's thesis in Nanotechnology
Supervisor: Astrid Aksnes
Co-supervisor: Arnfinn Aas Eielsen, Jens Høvik
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Abstract

The use of nanophotonics in biosensing is an interdisciplinary area of science that has gained significant attention and has evolved rapidly in recent years. The ability to integrate several photonic structures on a single semiconductor chip as a label-free optical biosensor is of particular interest. Due to advances in fabrication techniques in micro-and nanotechnology, improved miniaturization of such photonic devices yields more detailed structures, lower costs and lower energy consumption. The Mach-Zehnder interferometer-assisted ring resonator configuration (MARC) sensor is such an optical biosensor which can achieve a tailored spectral response, which can be refined to yield a low detection limit and a large dynamic range.

In this thesis, transmission signals of different MARC devices are examined. The current characterization setup has also been examined to find possible improvements in the data acquisition process, such as increasing the reproducibility and accuracy of each measurement. Furthermore, different signal processing approaches have been utilized to investigate the transmission curves and the accompanying noise. The sensing capabilities of the MARC sensor relies on the detection of resonance shifts occuring upon the presence of an analyte. A method has been developed through the course of this work for automatically calculating these resonance shifts.

Sammendrag

Bruken av nanofotonikk i anvendelsen av biosensorer er et tverrfaglig vitenskapsområde som har tiltrukket mye oppmerksomhet, samt utviklet seg hurtig i senere år. Muligheten å integrere flere fotoniske strukturer på en enkel halvlederbrikke til bruk av 'label-free' optiske biosensorer er av spesiell interesse. Forbedret miniatyrisering av slike enheter, grunnet fremskritt i fabrikasjonsteknikker i mikro- og nanoteknologi, gir mer detaljerte strukturer, lavere kostnader og lavere energiforbruk. Sensorer basert på en Mach-Zehnder interferometerassistert ringresonatorkonfigurasjon (MARC) er en slik optisk biosensor som kan oppnå en skreddersydd spektralrespons, som kan gi lav deteksjonsgrense og en høy dynamisk respons.

I denne avhandlingen vil transmisjonssignaler av ulike MARC-enheter bli undersøkt. I tillegg har det nåværende karakteriseringsoppsettet blitt undersøkt for å finne mulige forbedringer i datainnsamlingsprosessen, slik som en økning i reproduserbarheten og nøyaktigheten av hver måling. Videre har ulike fremgangsmåter i signalbehandling blitt brukt for å undersøke transmisjonskurvene og den medfølgende støyen. Sensoregenskapene til MARC-enhetene avhenger av deteksjonen av resonansskift som følge av tilstedeværelsen av en analytt. En metode har blitt utviklet i løpet av dette arbeidet for automatisk utregning av slike resonansskift.

Preface

This thesis was written as the concluding part of the degree of Master of Science in Nanotechnology offered by the Norwegian University of Science and Technology (NTNU). On submission, this work finalizes the five-year MTNANO study program with a specialization in nanoelectronics. The work presented in this thesis has been carried out under the supervision of Professor Astrid Aksnes at the Department of Electronic Systems (IES) during the spring semester of 2022. In addition, associate professor Arnfinn Aas Eielsen at the University of Stavanger (UiS) and Ph.D. candidate Jens Høvik at IES have acted as co-supervisors.

I want to thank my supervisor Astrid Aksnes for offering such an exciting project and for her advice and encouragement throughout the semester. I would also like to thank my co-supervisors Arnfinn Aas Eielsen and Jens Høvik, for their invaluable help and useful discussions about the subject.

I want to thank Mukesh Yadav and Jong Wook Noh for their help and training in the photonics laboratory and for always taking time off their work to assist me. I also want to thank Nanolab at NTNU and The Research Council of Norway is acknowledged for the support to the Norwegian Micro-and Nano-Fabrication Facility, NorFab, project number 295864.

On a final note, I want to thank my friends and family for their love and support. In particular, I want to thank Jonas Lyng-Jørgensen for helping me proofread this work.

[illegible]

Nikolai Stensø
Trondheim, June 2022

Contents

1 | Introduction

Throughout the last century, the life expectancy of humans has been steadily increasing due to an extensive health transition, in which medical technology has played an integral part [1]. As the medical field still advances at a tremendous pace, the field of biosensing and its capability to detect bacteria, viruses, and other toxins has become an essential tool in fighting disease and global pandemics [2–6].

A biosensor is a device that utilizes a biological recognition element in order to detect the presence of a chemical or biological species, called an analyte [2]. A subfield of the broad topic of biosensors is the interdisciplinary field of optical biosensing, where optical techniques are utilized for detecting and identifying analytes [2]. Besides the use mentioned above in the medical field, optical biosensors also find a wide range of applications in clinical diagnostics, drug development, environmental monitoring, and food quality control [2, 7, 8].

An optical biosensor operates by detecting a change in a physical, observable quantity such as intensity, phase, frequency, or polarization of light. To observe such a change a biological recognition element is utilized, which creates a response based on the presence of a species, or a physiological change [2]. This response is subsequently converted into a signal by using a transducer, which can be detected. By immobilizing the biorecognition element on an optical element, such as an optical fiber or a waveguide, the sensitivity and selectivity of the optical response can be significantly improved [2, 7]. The main components of an optical biosensor can be divided into five parts, which are summarized by

(i) a light source,

(ii) an optical transmission medium,

(iii) a biological recognition element,

(iv) a transducer, and

(v) an optical detection system.

The photonics group at NTNU is developing a specific type of optical biosensor, known as a Mach-Zehnder interferometer assisted ring resonator configuration (MARC).

The device is a lab-on-a-chip (LOC) device capable of measuring concentrations in various fluids [9]. As the name suggests, the sensor comprises microring cavities that act as resonators causing a phase shift of the incoming light. An interferometer is used to detect this phase shift physically. Additionally, the device allows for label-free biosensing, which helps overcome the problems of reliability and stability found in the detection of labeled molecules [10].

Thesis Objectives

In this thesis, the interest lies in component (v) of an optical biosensor, namely the optical detection system. This interest comprises not only the necessary experimental setup for detecting such physical changes in a system but also the subsequent data processing. The goal of this work is twofold; to outline the characterization techniques used for data collection of MARC devices and the development of an automated data processing technique that measures the observed change in intensity caused by a change in refractive index. Performing transmission measurements as a function of varying wavelengths on a MARC sensor yields a characteristic signature of the specific device. When the recognition element detects an analyte, the signature will experience a resonance shift of the corresponding ring resonator peaks. This work will explore a method of automatically calculating this resonance shift. Although the device allows for complex functionalizations and specific biological recognition elements, only bulk refractive index changes will be considered in this work.

Thesis Outline

In chapter 2, the fundamentals of light propagation in waveguides will be covered to the extent needed to understand the observed transmission signatures produced by MARC structures properly. Additionally, the chapter briefly introduces the different minimization algorithms utilized in this work. In chapter 3, an overview of the experimental setup used by the photonics group at NTNU will be given, in addition to an explanation of different software used and developed throughout this work. Chapter 4 contains the methods and procedures utilized to arrive at the final results, which will be given along with a discussion regarding their significance and validity. To conclude, a thesis summary will be provided in chapter 6.

2 | Theory

As the results of this work rely on an understanding of a range of different physical phenomena and digital algorithms, this chapter aims to give a fundamental review of the theory required in order to explain the results adequately. First, some basic theory of electromagnetic waves and waveguides will be presented such that the intensity output of a MARC sensor can be described mathematically. This part will closely follow the development of the material as done in the work by *Saleh* and *Teich* [11]. The main chapters of interest in this work are chapter 5, which concerns electromagnetic optics, and chapter 9, which concerns guided-wave optics. The last part of this chapter describes the minimization algorithms used in this work.

2.1 Electromagnetic Waves

The foundation of most advanced discussion about electromagnetic theory is Maxwell's equations. The general theory will be briefly discussed to clarify the notation used when discussing more advanced topics later on. As most of the discussion in this work concerns the propagation of light through waveguides, a natural starting point of the discussion is the equations for light traveling through a linear, nondispersive, homogeneous, isotropic and source-free medium. Using the notation $\boldsymbol{\mathcal{E}}(\mathbf{r}, t)$ for the complex electric vector field and $\boldsymbol{\mathcal{H}}(\mathbf{r}, t)$ for the complex magnetic vector field, Maxwell's equations are given by [11]

$$\nabla \times \boldsymbol{\mathcal{H}} = \epsilon \frac{\partial \boldsymbol{\mathcal{E}}}{\partial t} \tag{2.1}$$

$$\nabla \times \boldsymbol{\mathcal{E}} = -\mu \frac{\partial \boldsymbol{\mathcal{H}}}{\partial t} \tag{2.2}$$

$$\nabla \cdot \boldsymbol{\mathcal{E}} = 0 \tag{2.3}$$

$$\nabla \cdot \boldsymbol{\mathcal{H}} = 0. \tag{2.4}$$

In this context, $\epsilon = \epsilon_r \epsilon_0$ is called the *electric permittivity* of the medium and $\mu = \mu_r \mu_0$ the *magnetic permeability* of the medium. These two material properties are here described in terms of their relative value to their vacuum counterparts, denoted by the subscript 0. As the materials used in this work are assumed to be non-magnetic, the permeability of vacuum μ_0 is used.

2.1.1 Speed of Light and Refractive Index

The two properties ϵ and μ of a material can be used to define a constant known as the speed of the electric field in the given material, usually denoted by c. The constant determines the speed at which light travels in a dielectric medium and given by [11]

$$c = \frac{1}{\sqrt{\epsilon\mu}}. \tag{2.5}$$

It is also common to use the notation c_0 to describe the value in a vacuum. As with its material property constituents, it is useful to consider the ratio between the speed of light in a vacuum and the speed of light in a given medium. This ratio describes how the speed of light changes as it crosses a boundary between vacuum and the given material and is termed the *refractive index*. The refractive index is usually denoted by the symbol n and described mathematically as [11]

$$n = \frac{c_0}{c}. \tag{2.6}$$

A planar boundary between two materials can be considered, where the two materials have refractive indices n_1 and n_2, with $n_1 > n_2$. When an electromagnetic wave is incident on the boundary at an angle θ_1 in the material with refractive index n_1, the wave will refract when travelling across the boundary according to Snell's law, which is given by [11]

$$n_1 \sin\theta_1 = n_2 \sin\theta_2. \tag{2.7}$$

A special case occurs when θ_1 reaches a certain value θ_c, called the *critical angle*, in which θ_2 reaches 90° and refraction no longer occurs. The critical angle is given by [11]

$$\theta_c = \sin^{-1}\frac{n_2}{n_1}. \tag{2.8}$$

When this happens, a phenomenon called *total internal reflection* occurs in which all the light is reflected at the boundary and stays within the material with refractive index n_1. Light incident on the boundary with angles above, at, and below the critical angle are shown in figure 2.1. The complementary critical angle $\bar{\theta}_c = \pi/2 - \theta_c$ is also often used when discussing the total internal reflection.

2.1.2 Wave Equation and Helmholz Equation

By denoting any vector component of either $\boldsymbol{\mathcal{E}}$ or $\boldsymbol{\mathcal{H}}$ by u, it can be shown that all components satisfies the wave equation, which is given as [11]

$$\nabla^2 u - \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} = 0. \tag{2.9}$$

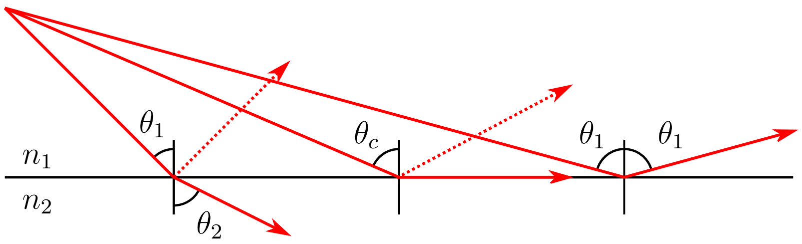


Figure 2.1: Illustration demonstrating total internal reflection. Assuming $n_1 > n_2$, light rays hitting the boundary between the two media are either refracted or reflected depending on the incident angle θ_1 of the ray. The dotted lines represent reflected waves when refraction of ray occurs. Adaptation of a figure found in [11]

If we consider electromagnetic waves that are *monochromatic*, i.e. of only one wavelength ν, the complex vector fields $\boldsymbol{\mathcal{E}}$ and $\boldsymbol{\mathcal{H}}$ can be described as harmonic functions as

$$\boldsymbol{\mathcal{E}}(\mathbf{r},t) = \mathrm{Re}\left\{\mathbf{E}(\mathbf{r})\exp(j\omega t)\right\} \tag{2.10}$$
$$\boldsymbol{\mathcal{H}}(\mathbf{r},t) = \mathrm{Re}\left\{\mathbf{H}(\mathbf{r})\exp(j\omega t)\right\}, \tag{2.11}$$

where $\omega = 2\pi\nu$ is the angular frequency corresponding to the frequency ν, and j is the imaginary unit. The vectors $\mathbf{E}$ and $\mathbf{H}$ describes the complex amplitude of their field counterparts. Inserting equations 2.10-2.11 into equation 2.9 yields the wave equation describing monochromatic waves, better known as *Helmholtz equation*. This is given as [11]

$$\nabla^2 U(\mathbf{r}) + k^2 U(\mathbf{r}) = 0. \tag{2.12}$$

Here the function $U(\mathbf{r})$ describes the complex amplitude of u, similar to what is done in equations 2.10-2.11, where u is the vector component of either $\boldsymbol{\mathcal{E}}$ or $\boldsymbol{\mathcal{H}}$. The constant $k = \omega\sqrt{\epsilon\mu} = nk_0$ is called the *angular wavenumber*, or more commonly just the wavenumber [11]. The quantity $k_0 = 2\pi\nu/c_0 = \frac{2\pi}{\lambda_0}$ is the vacuum wavenumber, while λ_0 is the wavelength of the electrimagnetic wave in vacuum. Another concept related to the wavenumber is the vector quantity known as the *wavevector* $\mathbf{k}$. The wavevector determines the direction of propagation of the electromagnetic waves and has the property $|\mathbf{k}| = k$ [11].

2.1.3 Polarization

For an electromagnetic wave $\boldsymbol{\mathcal{E}}$ propagating in time t, the curve traced out by the amplitude of the field vector at a given position in space $\mathbf{r}$ is called the *polarization* of light. If we consider a monochromatic plane wave as described in equation 2.10, the two orthogonal vector components of $\mathbf{E}$, which in general differ in both amplitude and phase, will draw out an ellipse characterizing the wave as an *elliptically polarized* wave [11]. Depending on the amplitudes and phases of the vector components, this ellipse may reduce to a circle or a straight line.

Waves propagating along the optical axis of the wave are approximated as *transverse electromagnetic* (TEM) waves, where the propagation direction, electric field, and magnetic field are all perpendicular to each other. TEM waves are modeled as plane waves, where only a single polarization ellipse describes the polarization of the wave. The orientation and ellipticity of the ellipse determine the polarization state, while the size of the ellipse determines the optical intensity [11].

2.1.4 Dispersion

Some dielectric materials are characterized by a frequency-dependent behavior of some of its material properties, such as the electric permittivity $\epsilon(\nu)$, the refractive index $n(\nu)$ and thus the speed $c(\nu) = c_0/n(\nu)$ [11]. As the wavelength λ is related to the frequency ν by $\lambda = c/\nu$, dispersive media changes its behaviour depending on the wavelength of the incoming light as well. An empirical model of the refractive index $n(\lambda)$ of a dispersive material depending on the wavelength λ of the incoming light is the *Sellmeier equation*, which is given by [11]

$$n^2(\lambda) = 1 + \sum_i \frac{B_i \lambda^2}{\lambda^2 - C_i}. \tag{2.13}$$

The coefficients B_i and C_i for $i = 1, 2, \ldots$ are called the *Sellmeier coefficients*. Although the model allows for an arbitrary high value for i, it is often enough to use $i = 1$ to get a good enough approximation.

2.1.5 Waveguides

An *optical waveguide* is a structure that can confine electromagnetic waves and carry the light over some distance [11]. The basic principle for this confinement is that of total internal reflection, as described previously. Although many types of waveguides exist, the one of interest in this work is a type that utilizes a dielectric material for the confinement of light.

Electromagnetic waves traveling in a waveguide cannot be TEM waves due to the boundary conditions imposed on Maxwell's equations [11]. Instead, the wave propagating through a waveguide can be either *transverse electric* (TE) or *transverse magnetic* (TM), depending on whether the electric or magnetic field is transverse to the direction of propagation. As the waveguides used in this project are constructed for the propagation of TE waves, the following theory on wave propagation in waveguides will only consider these.

Surrounding a dielectric material of width d and refractive index n_1 by other media of some lower refractive index n_2 produces what is called a *planar dielectric* waveguide [11]. In such a waveguide, the confined light is guided in the z-direction, with light rays making angles θ in the $y - z$ plane, as shown in figure 2.2.

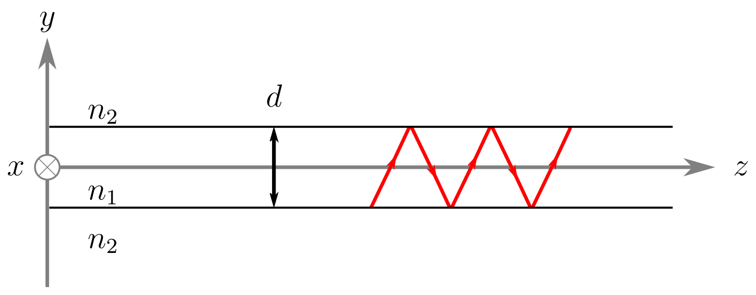


Figure 2.2: An illustration of wave propagation in a dielectric waveguide. The red line illustrates the beam, which oscillates along the z-axis between two media of refractive index n_2. The x-axis goes into the plane. The length d is the height of the dielectric medium of refractive index n_1.

Modes

A monochromatic TE wave of wavelength $\lambda = \lambda_0/n_1$, wavenumber $k = n_1 k_0$, wavevector $\mathbf{k} = [k_x, k_y, k_z]^T = [0, n_1 k_0 \sin\theta, n_1 k_0 \cos\theta]^T$, phase velocity $c = c_0/n_1$ and the electric field in the x-axis will propagate through the dielectric slab by oscillating between the upper and lower boundaries of the material [11]. As the wave bounces from boundary to boundary in the waveguide, a phase shift is introduced to the wave equal to $(2\pi/\lambda)2d\sin\theta$ after bouncing off the boundaries twice. This phase shift arises because the oscillating wave has traveled a distance $2d\sin\theta$ further than a wave only traveling along the z-axis. In addition, there is a second phase contribution φ_r caused by each internal reflection at the dielectric boundaries [11]. For a wave to maintain propagation thoughout the waveguide, it needs to reproduce itself after two reflections. This is to avoid the destructive interference caused by the phase difference [11]. This condition imposed on the waves is called the *self-cconsistency condition* [11]. Reproduction means that the phase must be an integer multiple m of 2π after two reflections, i.e.

$$\frac{2\pi}{\lambda}2d\sin\theta - 2\varphi_r = 2\pi m. \tag{2.14}$$

The fields that satisfy the self-consistency conditions are called the *modes* of the waveguide [11]. The modes of a waveguide maintain the same transverse distribution and polarization along the whole waveguide. The phase shift φ_r depends on the angle θ and whether the polarization of the incident wave is TE or TM. Solving equation 2.14 for θ yields a set of discrete bounce angles $\theta_m \in (0, \bar{\theta}_c)$ for which the wave can propagate [11].

These angles have corresponding wavevectors $\mathbf{k}_m = [0, n_1 k_0 \sin\theta_m, n_1 k_0 \cos\theta_m]^T$, where each component corresponds to the propagation constant for that direction. Since the wave is travelling in the z-direction, the z-component of $\mathbf{k}_m$ gives rise to an *effective refractive index* for mode m, given by [11]

$$n_{\text{eff,m}} = n_1 k_0 \cos\theta_m, \tag{2.15}$$

which the propagating wave is experiencing as it travels through the waveguide.

When the waveguide is sufficiently thin, only one mode can propagate through the waveguide. In this case, the waveguide is called *single-mode* [11]. The effective refractive index is then given as n_{eff}.

Coupling

When two waveguides are sufficiently close to one another, the electromagnetic radiation can be coupled, i.e. the radiation travels from one waveguide to the other [11]. An approximation known as coupled-mode theory is used to describe the phenomenon of weak coupling. This theory assumes that the mode of each waveguide is calculated as if the other waveguide was absent. The coupling is assumed only to affect the amplitudes of the modes [11]. In order to make the exposition given here more clear, some further assumptions are made. Namely that the waveguides are *phase matched*, i.e. have identical material properties, and that they are single-mode. By making these assumptions, the input and output amplitude fields are related by the transmission matrix T, such that the output electric field $\mathbf{E}_{\text{o}}$ is given by [11]

$$\mathbf{E}_{\text{o}} = T\mathbf{E}_{\text{i}}, \tag{2.16}$$

where $\mathbf{E}_{\text{i}}$ denotes the input field. The fields $\mathbf{E}_{\text{o}}$ and $\mathbf{E}_{\text{i}}$ correspond to the fields at the waveguides marked by *output* and *input* in figure 2.3, respectively. The transmission matrix is given by [11]

$$T = \begin{bmatrix} A(z) & B(z) \\ C(z) & D(z) \end{bmatrix} = \begin{bmatrix} \cos \mathcal{C}z & -j\sin \mathcal{C}z \\ -j\sin \mathcal{C}z & \cos \mathcal{C}z \end{bmatrix}, \tag{2.17}$$

where the constant $\mathcal{C}$ is called the coupling coefficient. This coefficient determines how much power is transferred while the waveguides are close to one another. The *coupling length* $L_0 = \pi/2\mathcal{C}$ is the length at which all the power is transferred from one waveguide to the other [11]. At a length $L_0/2$, half the power is transferred. Such a coupling is called a 3 dB coupler and is shown in figure 2.3.

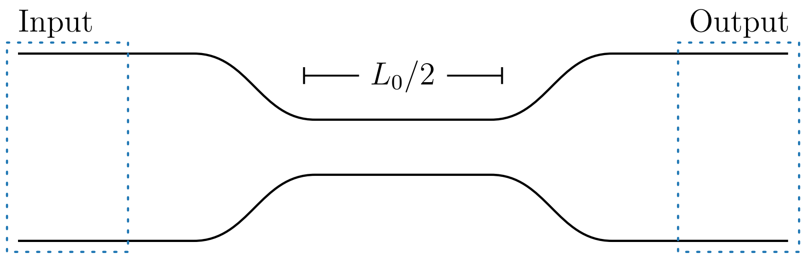


Figure 2.3: A schematic of a 3 dB coupler, where the two lines illustrate two different waveguides. The blue rectangles indicate the input and output ports of the waveguides. The waveguide coupling occurs in the area in the middle, where the waveguides are close to each other. The coupling length is marked as $L_0/2$, which defines the 3 dB coupler.

2.1.6 Intensity and Interference

It is often useful to consider the energy related to a harmonic wave U, as this energy is a physical quantity that can be measured. A parameter known as the *intensity* I of such a wave can be defined as the time average of the amount of energy that crosses a unit area, where the area is perpendicular to the direction of energy flow of the wave [12]. For a plane wave, this quantity can be given as

$$I = \frac{c}{4\pi}\sqrt{\frac{\epsilon}{\mu}}\langle U^2\rangle, \tag{2.18}$$

where $\langle . \rangle$ denotes the time average over some interval. This quantity can be simplified by modifying the scaling constant of the amplitude-dependent term, such that the intensity is expressed instead as $2\langle U^2\rangle$ [11]. Although the calculated value will not be the same using the different scaling constants, the expression is often useful when the exact value of the intensity is not of interest, such as when comparing different intensities against one another. The expression can be further simplified if the average is taken over a time interval which is large compared to the period T of the wave, in which case the intensity can be expressed as $I = 2\langle U^2\rangle = |U|^2$[11, 12].

Superposition of electromagnetic waves follows from the linearity of the wave equation, meaning that the total wavefunction of a wave comprising two or more waves in the same region will be the sum of the individual wavefunctions [11]. However, this principle does not apply to intensity, meaning that the sum of two or more waves does not necessarily equal the sum of their individual intensities [11, 12]. The lack of this principle in intensity gives rise to the concept of *interference*, which describes this phenomenon [11].

Suppose a wave $U(\mathbf{r}) = U_1(\mathbf{r}) + U_2(\mathbf{r})$ comprises two superposed monochromatic waves with complex amplitudes $U_1(\mathbf{r})$ and $U_2(\mathbf{r})$. The resulting wave U will be monochromatic with the same frequency as the two waves it is composed of. Using the notation $U_i = \sqrt{I_i}\exp\{(j\varphi_i)\}$ where $i \in \{1, 2\}$, the intensity of this wave can be described by the *interference equation*, which is given by [11]

$$\begin{aligned} I &= |U_1 + U_2|^2 \\ &= I_1 + I_2 + 2\sqrt{I_1 I_2}\cos\varphi, \end{aligned} \tag{2.19}$$

where $\varphi = \varphi_2 - \varphi_1$ describes the phase difference between the two waves. From equation 2.19 it is apparent that the intensity of a wave comprising two waves is not equal to the sum of the intensities of the two waves, as the expression contains a phase-dependent term. It is this term which describes the interference of the two waves, and is a measure of the correlation between the two different waves [12]. In general will two waves originating from the same source be correlated, meaning interference will be observed of the superposed wave. For two waves originating from different sources, the superposed wave will in general not exhibit interference.

An *interferometer* is a device that utilizes the phenomenon of interference by splitting a wave into two waves using a beamsplitter, changing the phase of one of the two split waves, and subsequently recombining the two waves back into one [11]. The intensity of the superposed waves at the output is then detected. Using such a device makes it possible to experimentally detect the phase difference induced on one of the waves. A basic illustration of the operation of an interferometer is shown in figure 2.4 The phase difference can be induced by various methods, such as different optical path lengths or different propagation directions [11]

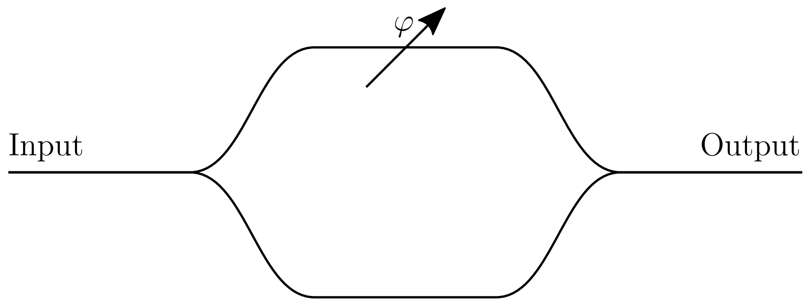


Figure 2.4: A basic illustration of an interferometer. An incoming wave enters the interferometer at the input and is split into two waves by a beamsplitter. The wave in one of the arms is subject to a phase shift φ before the waves are superposed together before detection at the output.

2.1.7 Ring Resonators

An *optical resonator* is a device that is capable of accumulating and confining light of specific *resonant* frequencies, which are determined by the different parameters of the device [11]. There is a large variety of different configurations which enable this capability, such as the *planar-mirror*, *rectangular-cavity*, and *photonic-crystal* resonator configurations [11]. In this text, only the ring resonator will be discussed in detail.

A *ring resonator* is a circular waveguide capable of sustaining resonating modes. While not very interesting on its own, the ring can be coupled with other straight waveguides to couple light in and out of the ring. The simplest such structure consists of only a single ring and a single waveguide and is known as a *all-pass* ring resonator [13]. In order to describe the input and output electric fields of the structure, some basic assumptions have to be made. These assumptions are that: [13]

- only a single unidirectional mode of the resonator is excited,
- the coupling is lossless,
- only single polarization is considered,

- the different waveguide segments and coupler elements do not couple waves of different polarizations,
- the attenuation constant contains all the propagation losses through the ring resonator.

As long as these assumptions hold, the electric field can be described quite similarly as for two coupled waveguides, which is described in equation 2.16. For the structure containing only one ring and one waveguide, the coupling is described by [13]

$$\begin{bmatrix} E_{t1} \\ E_{t2} \end{bmatrix} = \begin{bmatrix} t & \kappa \\ -\kappa^* & t^* \end{bmatrix} \begin{bmatrix} E_{i1} \\ E_{i2} \end{bmatrix}, \tag{2.20}$$

where t and κ are the coupler parameters, which are required to satisfy $|\kappa^2| + |t^2| = 1$ because of the symmetry of the matrix [13].

For a given ring of radius r with a light of period T passing through it, the light will experience a phase shift $\varphi = 2\pi\tau/T$, where $0 \leq \tau < T$ is the time spent in the ring. This time is found by dividing the circumference of the ring, $2\pi r$, by the speed at which the wave travels $c = c_0/n_{\text{eff}}$, where c_0 is the speed of light in vacuum and n_{eff} is the effective refractive index. Relating the period T of a wave with the wavelength yields the relation $T = \lambda/c_0$. Combining all this gives the total phase shift for a round trip in the ring as [13]

$$\varphi = 4\pi^2 n_{\text{eff}} \frac{r}{\lambda}. \tag{2.21}$$

Parallel Add-Drop Ring Resonators

Adding another waveguide to the all-pass ring resonator yields what is called an *add-drop* ring resonator, which is shown in figure 2.5 [13]. Here the names of the input and output ports are shown, namely the add, drop, throughput, and input ports.

To simplify the model of the output electromagnetic field of the ring it is assumed that $E_{i1} = 1$. It can then be shown that the electric field at the drop port is given by [13]

$$E_{t_2} = \frac{-\kappa_1^* \kappa_2 \alpha_{1/2} e^{j\varphi_{1/2}}}{1 - t_1^* t_2^* \alpha e^{j\varphi}}. \tag{2.22}$$

The parameters $\alpha_{1/2}$ and $\varphi_{1/2}$ are used to denote the change in loss and phase after a half round trip around the ring, respectively. To achieve a maximum intensity at the resonance wavelengths, the coupling parameters t_1 and t_2 need to be adjusted so that [13]

$$\alpha = \left| \frac{t_1}{t_2} \right|. \tag{2.23}$$

. This phenomenom is called *critical coupling* [13]. The intensity output at the drop port can then be calculated using $I_{t2} = |E_{t2}|^2$, which is shown under critical coupling

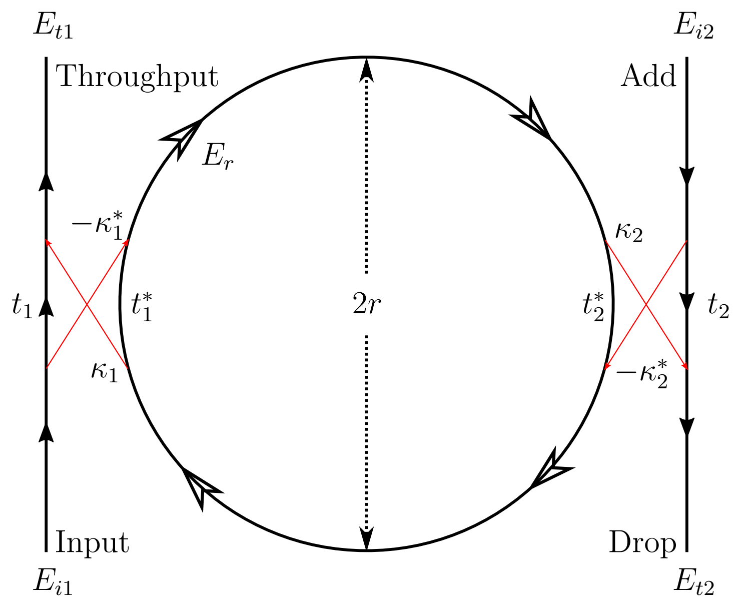


Figure 2.5: Illustration of an add-drop ring resonator of radius r. The black lines illustrate waveguides, with the arrows indicating the propagation direction. The red lines indicate the waveguide coupling. The notation used for the electric field at different positions throughout the structure is shown, in addition to the names used for the various waveguide ports. The illustration is based on a figure in [13].

in figure 2.6 for a ring of radius 25 µm.The intensity of a resonator is recognized by its distinct periodic peaks, which are separated by a wavelength distance known as the *free spectral range* (FSR) [11]. For a ring resonator with circumference $L = 2\pi r$ and a resonance peak at the vacuum wavelength λ_0 the FSR can be approximated as [13]

$$\mathrm{FSR} \approx \frac{\lambda_0^2}{n_{\mathrm{eff}} L}. \tag{2.24}$$

Non-Parallel Add-Drop Ring Resonators

The two waveguides coupling with the ring in an add-drop ring resonator need not be parallel. They can be separated by an arbitrary angular separation θ, as shown in figure 2.7. The amplitude transmission at the drop port can then be given as [14]

$$E_{t2} = -\frac{\sqrt{1-t_1^2}\sqrt{1-t_2^2}\alpha_\theta \exp\{j\varphi\frac{\theta}{2\pi}\}}{1 - t_1 t_2 \alpha \exp\{j\varphi\}}. \tag{2.25}$$

The round-trip phase shift is the same as given in equation 2.21, while α_θ is the fraction of the round-trip amplitude transmission factor between the input and output coupler [14].

The main difference between the different configurations created by using different

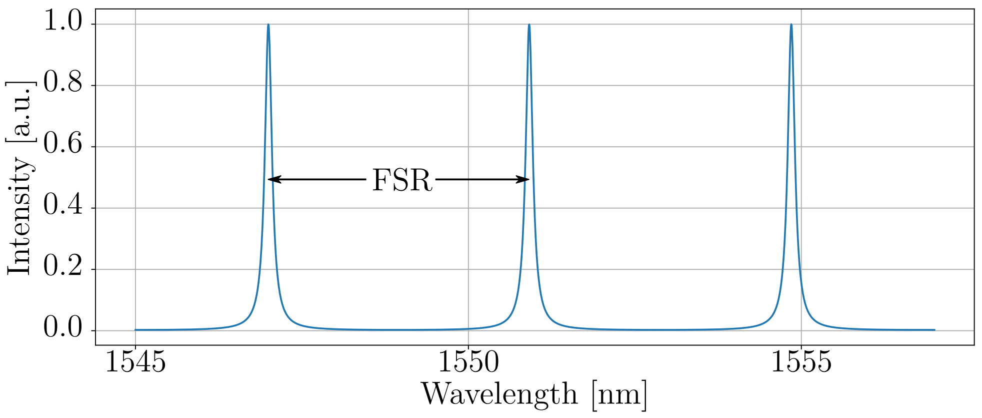


Figure 2.6: Normalized drop port transmission for a add-drop ring resonator with a ring of radius 25 µm and angular separation 180°. The wavelength distance corresponding to the FSR is indicated.

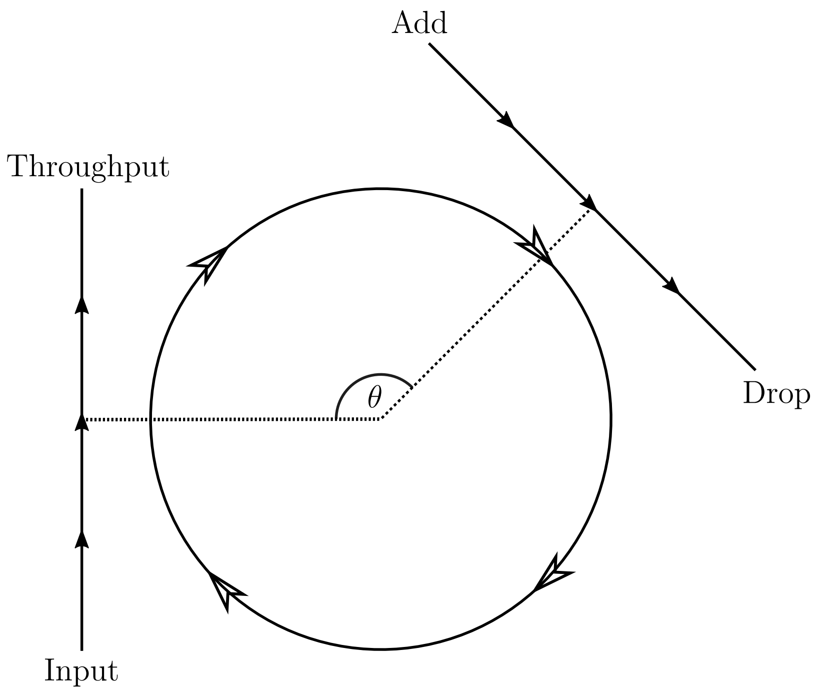


Figure 2.7: Illustration of a non-parallel ring resonator with angular separation θ. The arrows indicate the propagation direction of light in the waveguides. The special cases for this configuration are when $\theta = 0°$ or $\theta = 180°$. In these cases the structure is an all-pass or a parallel add-drop ring resonator, respectively.

angular separations is the difference in phase accumulation between resonances. For a parallel add-drop ring resonator, i.e. $\theta = 180°$, this phase changes monotonically as a function of wavelength, where the phase accumulation between resonances equals π. This accumulated phase difference is reduced in accordance with the angular separation angle reduction. The phase of the light transmitted at the drop port is shown in figure 2.8 and is described matematically as [14]

$$\phi = \arctan\frac{\text{Im}\{E_{t2}\}}{\text{Re}\{E_{t2}\}} = \pi + \varphi\frac{\theta}{2\pi} + \arctan\frac{t_1 t_2 \sin\varphi}{1 - t_1 t_2 \alpha \cos\varphi}. \tag{2.26}$$

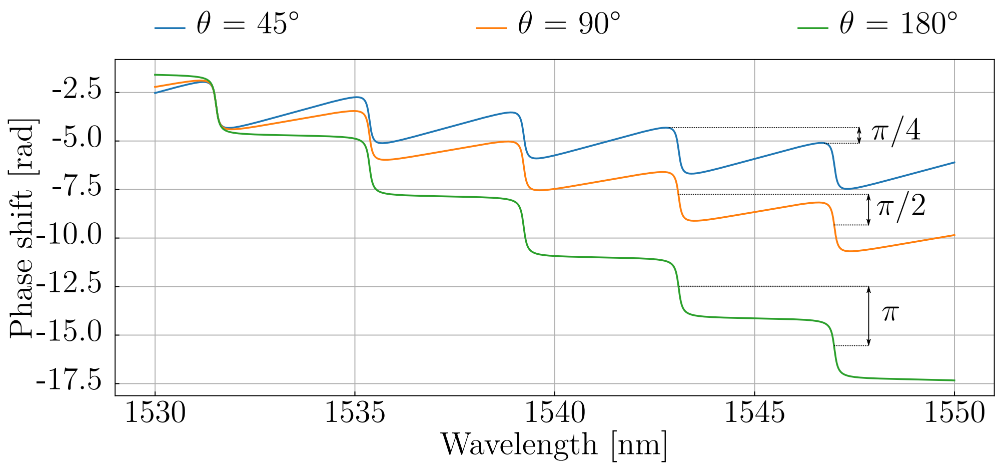


Figure 2.8: A plot showing the phase at the drop port of three different add-drop ring resonators of angular separations 45°, 90° and 180°.

2.2 MARC Devices

A *Mach-Zehnder interferometer assisted ring resonator configuration* (MARC) is a device combining a Mach-Zehnder interferometer with a ring resonator by placing the ring resonator in the phase-shifting arm of the interferometer [8, 14]. The intensity output detected will change at different wavelengths depending on the angular separation of the ring resonator drop port with respect to the throughput, as shown in figure 2.8. A simple schematic of a MARC device with angular separation 90° is shown in figure 2.9

In a simplified situation with only a single ring resonator, which is lossless ($\alpha = 1$) and critically coupled ($t_1 = t_2$), the intensity of such a device can be found using the interference equation 2.19 equation. By assuming the intensity I_1 in the unaffected arm to be one, and that the intensity in the phase shifted arm is given as $|E_{t2}|^2$, as derived in equation 2.25, the final expression for the transmitted light at the drop port can be written as

$$I = \frac{1}{4}(1 + |E_{t2}|^2 + 2|E_{t2}|\cos\phi), \tag{2.27}$$

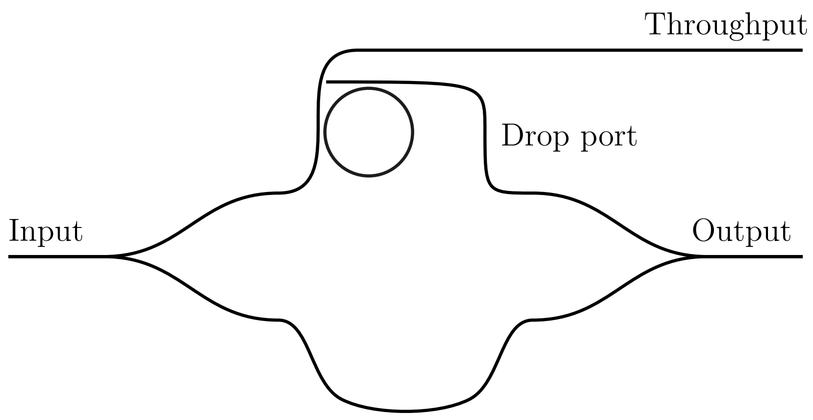


Figure 2.9: Schematic of a MARC device with one ring of angular separation 90°

The phase shift φ is assumed to be the phase shift caused by the ring resonator configuration as derived in equation 2.26. Additionally, the scaling factor 1/4 has been added to ensure that the maximum intensity of the expression is 1. A plot of such a transmission output across a range of wavelengths is shown in figure 2.10

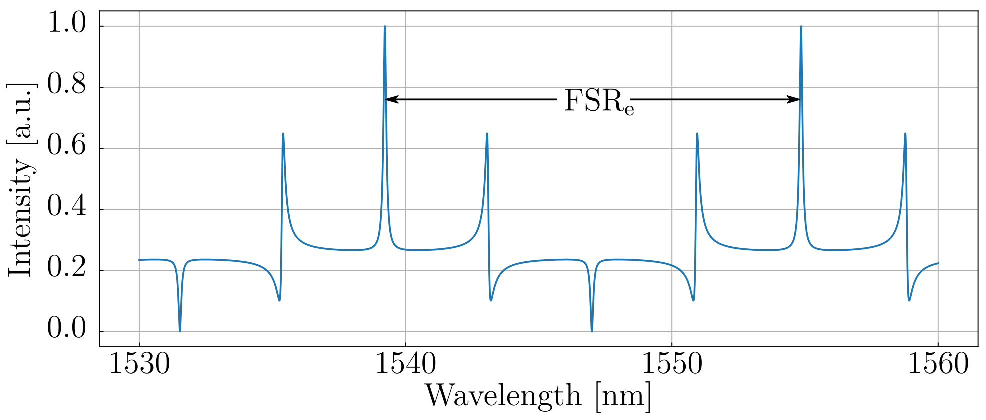


Figure 2.10: Drop port transmission output of a MARC with 90° angular separation. The intensity is given in normalized units. The FSR$_e$ is indicated for this particular MARC device.

2.2.1 Spectral Line Shapes

As can be seen from figure 2.10, a unique signature is created when sweeping across a wavelength range for a MARC of a specific ring radius and angular separation of the ring. Although the familiar symmetrical *Lorentzian* line shape, which is common in resonators, and the *inverse* Lorentzian can be recognized, other more unusual, asymmetric line shapes can be seen as well. This asymmetric line shape is known as a *Fano* resonance line shape [15–17]. This asymmetric line shape is not seen in conventional resonators, as it occurs when a discrete quantum state interferes with a continuum band of states [18]. In photonics, this can be caused by scattering involving two channels, where one of the channels is nonresonant broadband, and the other is resonant narrowband [19]. An example of such a system is the MARC sensor, which is a type of side-coupled waveguide-resonator system [15].

In fact, it turns out that both the Lorentzian and inverse Lorentzian line shapes are just special cases of the Fano line shape, depending on the asymmetry factor q [18, 20]. The Fano line shape is given by

$$\sigma(\omega) = D^2 \frac{(q+\Omega)^2}{1+\Omega^2}. \tag{2.28}$$

Here, ω is the angular frequency, $D = 2\sin(\phi/2)$ is a scaling factor, $q = \cot(\phi/2)$ is the Fano asymmetry parameter, ϕ is the phase shift, $\Omega = 2(\omega - \omega)/\gamma$ is the reduced frequency, where γ and ω_0 are the resonance width and frequency, respectively [18, 19]. The phase scaling factor 1/2 is added to accommodate the periodicities for both the phase shift and the cotangent function [20]. As the phase shift ϕ varies, the asymmetry parameter changes, which causes the apparent line shape to change as well. As ϕ reaches a multiple of 2π, the asymmetry parameter reaches either $+\infty$ or $-\infty$, and the line shape becomes Lorentzian, as shown in figure 2.11 (a). In figure 2.11 (b), the characteristic asymmetric Fano line shape can be seen, which occurs when the asymmetry parameter reaches 1, i.e. the phase shift equals $\pi/2$. Lastly, the inverse Lorentzian line shape, shown in figure 2.11, occurs when the phase shift equals π, which yields an asymmetry parameter of 0. As the cotangent function has a period of π, these line shapes repeat as the phase shift increases. For the values of ϕ between all those stated, other line shapes will manifest, which can be interpreted as intermediate states between the ones discussed.

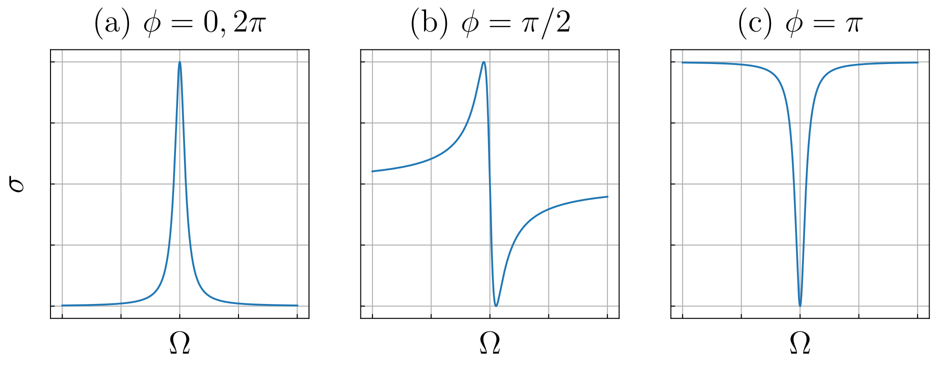


Figure 2.11: Plots illustrating different line shapes produced by equation 2.28, with the corresponding phase shifts ϕ within the period. (a) Lorentzian line shape, which occurs when the phase shift $\phi = 0$ or $\phi = 2\pi$. (b) The characteristic asymmetric Fano line shape, which occurs when $\phi = \pi/2$. (c) The inverse Lorentzian line shape, which occurs when $\phi = \pi$. All three line shapes are shown as the scattering cross-section σ as a function of the reduced frequency Ω.

To exemplify; the wavelength-dependent phase shift of a 90° ring resonator shown in figure 2.8 can be related to the transmission output of a MARC with angular separation 90°, as shown in figure 2.10. As the phase accumulates step-wise, with each step increasing by $\pi/2$, the MARC signature exhibits peaks corresponding to the same steps at each peak. A Lorentzian line shape, given by $\phi = 0$, is followed by an asymmetrical Fano line shape, which occurs when $\phi = \pi/2$ is again followed by an inverse Lorentzian, and so on.

These different line shapes result in the unique MARC signatures, which will repeat when the accumulated phase difference reaches an integer multiple of 2π . The period of the repeating transmission spectrum is defined as the *effective free spectral range* FSR_e [14], and is indicated for a 90° MARC in figure 2.10. As the name suggests, it is related to the FSR of the ring resonator in the device. To explain this relation a parameter L needs to be introduced. The parameter is calculated as $L = 2\pi/\theta$, where θ is the angular separation of the MARC given in radians [8]. There are two possibilities for the resulting value of L, depending on the angular separation of the MARC, where L is either a rational number or a positive integer. In the case where L is a rational number, the reduced fraction of L can be expressed as N/M, for some positive integers N and M. In the case L is a positive integer, the integer N is given as $N = L$[14]. The relation of FSR and FSR_e can then finally be expressed as [14]

$$\text{FSR}_\text{e} = N \cdot \text{FSR}. \tag{2.29}$$

2.2.2 Parameter Effect on Signature

As many parameters affect the final transmission response of the MARC device, there are many ways in which the signature can be affected. A general resonance shift of the signature may be caused by different factors, such as temperature change [14, 21, 22] or an externally applied electric field [23].

As the radius r of the ring resonator varies, the FSR of the ring will experience a change according to 2.24. This means that a smaller ring radius will result in a larger FSR, as shown in figure 2.12. By varying the angular separation θ of the ring resonator in the MARC, the resonance peaks will stay in the same positions, but the line shapes of the resonance peaks will change according to 2.28, as can be seen in figure 2.12.

Changes in the loss coefficient α will result in a vertical shift of the whole signature, as seen in figure 2.13. Likewise, variations in the coupler parameter t_1 will result in changes in the sharpness of the resonance peaks, as seen in the middle graph of figure 2.13. Lastly, a change in the effective refractive index of the ring results in both a horizontal shift of the signature and a change in the FSR, as seen in the bottom graph of figure 2.13. This results from the effective refractive index dependency of equation 2.24.

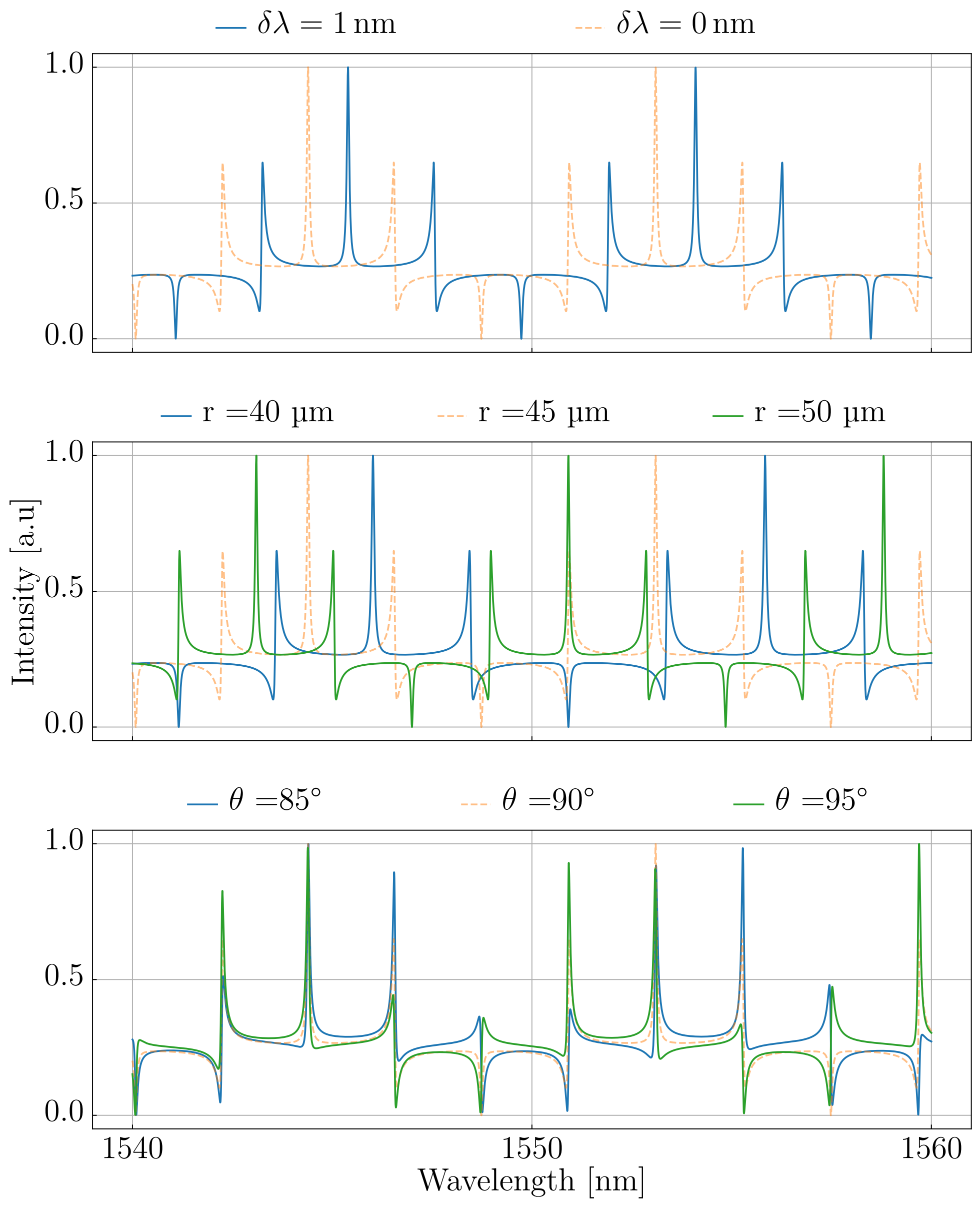


Figure 2.12: Overview of the effects on the signature caused by adjusting different paramters. The top plot shows a general resonance shift, which may be caused by different factors. The middle plot shows the effects of variations of the ring radius. The bottom plot shows the effects of variations of the angular separation of the ring resonator.

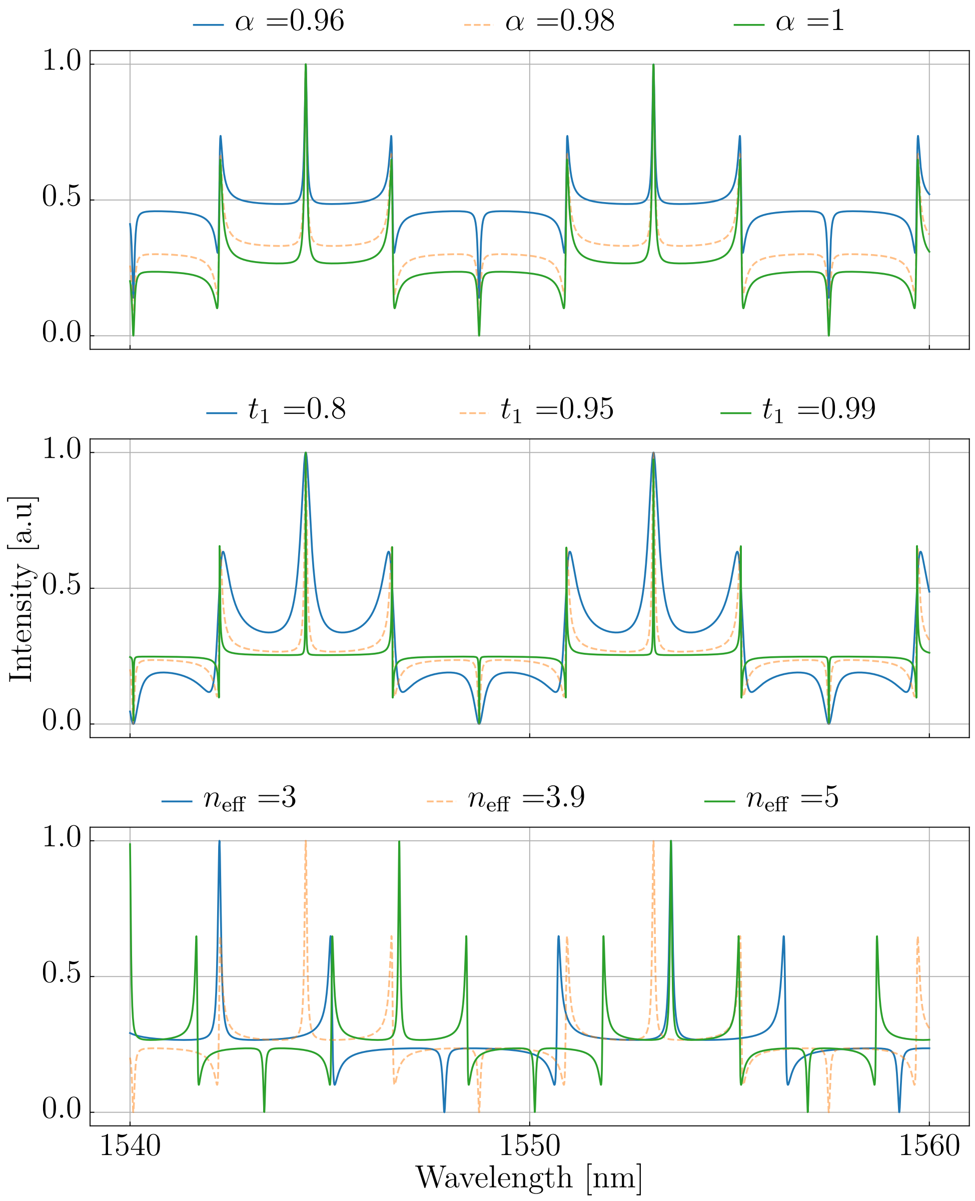


Figure 2.13: Overview of the effects on the signature caused by adjusting different parameters. The top plot shows a the effects of the loss coefficient α. The middle plot shows the effects of variations of coupler parameter t_1. The bottom plot shows the effects of variations of the effective refractive index.

2.2.3 MARC Devices as Sensors

Utilizing a MARC as a sensor involves exposing the ring resonators of the structures to some fluid containing the analyte to be detected. This exposure is performed for the effective refractive index to change, resulting in a phase difference [24]. As mentioned in chapter 1, this work will focus on bulk refractive index changes of MARC sensors, which relies only on a change in the refractive index of the fluid, and not on detecting specific analytes present. The effective refractive index of the ring resonator will then be sensitive to a bulk refractive index change in the surrounding fluid, which will result in a resonance shift of the transmission signature [8]. The total shift of the transmission signature, which is also known as the *sensitivity*, is given as [8]

$$S = \frac{\Delta\lambda}{\Delta n} = \frac{m}{L}\left(\frac{\partial n_{\text{eff}}}{\partial n_{\text{bulk}}}\right)^{-1}, \tag{2.30}$$

where L is the circumference of the ring resonator, m is the cavity mode order, n_{eff} is the effective refractive index of the guided mode and n_{bulk} is the bulk refractive index change of the fluid [8]. Another parameter to consider of a sensor is the measurement range Δn_{max}, which is the highest detectable change in refractive index, and is given as

$$\Delta n_{\text{max}} = \frac{\text{FSR}_{\text{e}}}{S}. \tag{2.31}$$

2.3 Optimization Algorithms

Mathematical optimization is the process of minimizing a scalar *objective function* with respect to its *design variables*, which is normally given as a vector [25]. As the objective function is calculated based on its design variables, a minimum of the function is found when the optimal design variables are found. The method of acquiring these optimal variables is usually based on an algorithm that iteratively minimizes the objective function value [25]. The algorithm ends when some *optimality criterion* has been satisfied. There are many different minimization algorithms, which all differ in how they operate and what problems they solve. One way of bisecting such algorithms is whether they are mathematical or heuristic. Mathematical algorithms are often gradient-based and use some mathematical principle for the optimality principle [25]. This means that the algorithm ensures that the objective function is at a minimum when the optimal variables have been found. On the other hand, heuristic algorithms will often not require the gradient of the objective function. However, the optimality criteria for heuristic algorithms do not ensure that the optimal variables found give an actual minimum of the objective function, but rather a point that is "close enough" [25].

2.3.1 Data Fitting

Given a set of n observations, where each observation consists of an independent variable X_i and a dependent variable Y_i, where $i = 1, \ldots, n$, it is often of interest to fit the data points to a model. In general, such a model can be represented by finding a *fitted value* y_i for observation Y_i which is an approximation of the observation value, given by $y_i = Y_i + r_i$, where r_i is called the *residual* [26]. In practice, the fitted value y_i is often calculated from some specific function $y(X_i, \mathbf{x})$, which depends on the independent variable X_i in addition to an arbitrary number of other parameters $\mathbf{x}$, often represented as a vector. The task of finding the best curve fit of of the points $y_i = y(X_i, \mathbf{x})$ to the observations Y_i then turns into minimizing the residuals r_i, such that the values y_i and Y_i are as close to each other as possible. The residuals can be represented as [26]

$$r_i(\mathbf{x}) = Y_i - y(X_i, \mathbf{x}). \tag{2.32}$$

The residuals are used to construct an objective function, such that a minimization algorithm can be used for optimizing the curve fit. In this work, two different minimization algorithms are explored for such use, namely the Nelder-Mead minimization algorithm and non-linear least squares regression. The Nelder-Mead algorithm is a heuristic algorithm, while least squares is a mathematical one [25]. Although the derivative can not be found for the residuals, they can be approximated.

2.3.2 Nelder-Mead

One of the most common derivative-free optimization algorithms for solving unconstrained problems is the Nelder-Mead algorithm [27]. In this algorithm, a function $f : \mathbb{R}^n \to \mathbb{R}$ called the objective function (also called a cost function) is minimized by generating a sequence of *simplicies* and iteratively approximates the minimum of f [25, 27]. A simplex, denoted Δ, is a geometric structure which is the convex hull[1] of $n+1$ vertices $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n+1} \in \mathbb{R}^n$ in n dimensions[27]. To provide a visualization of simplices it should be noted that in one dimension a simplex is a line, and in two dimensions the simplex is a triangle [25].

For each iteration of the algorithm a simplex is iteratively generated using four possible operations called *reflection*, *expansion*, *contraction* and *shrinking* [27]. These operations are associated with the scalar parameters $\alpha > 0, \beta > 1, 0 < \gamma < 1$ and $0 < \delta < 1$, respectively. The values commonly used for the simplex operation parameters are[27]

$$[\alpha,\ \beta,\ \gamma,\ \delta] = [1,\ 2,\ 0.5,\ 0.5]. \tag{2.33}$$

To perform the different simplex operations, the vertices of the simplex first needs to be ordered from what is called the *best* vertex $\mathbf{x}_1$ to the *worst* vertex $\mathbf{x}_{n+1}$ [27].

[1]A subset of $\mathbb{R}$ is convex if it contains the straight line segment between any two points in the subset [28]. A convex hull of a set of points is the intersection of all convex sets containing the set.

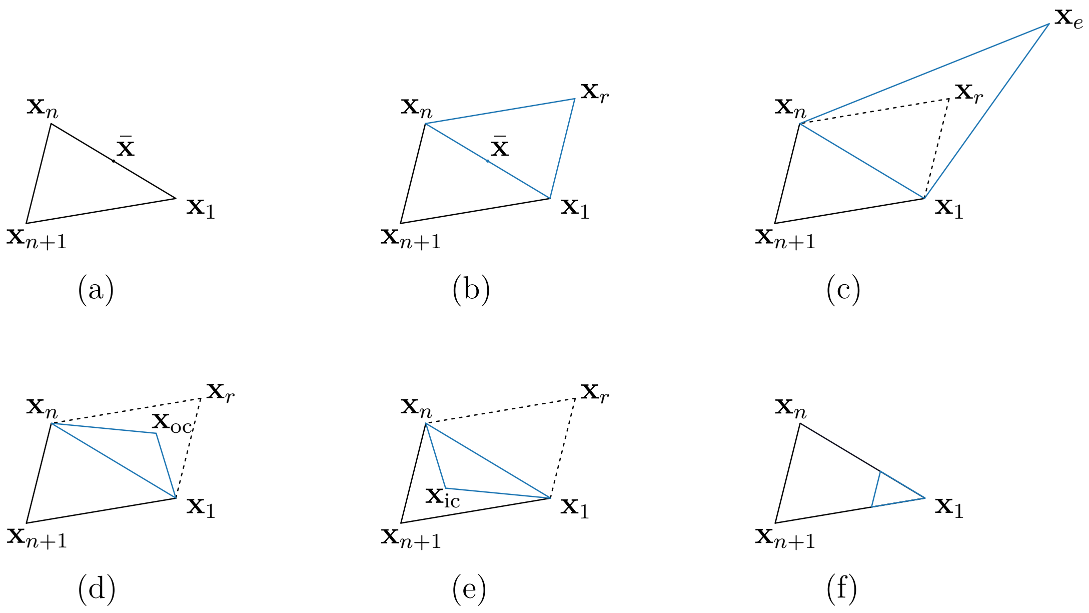


Figure 2.14: Illustration of the $n = 2$ simplex and the simplex operations. (a) Initial simplex (b) reflection, (c) expansion, (d) outside contraction, (e) inside contraction and (f) shrinking. Figure adapted from [25].

The idea behind this is to determine which vertex yields the lowest function value, in order to replace it with a more optimal vertex. The ordering is thus found by calculating the function value of every vertex of the simplex and ordering them as follows [27]:

$$f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \cdots \leq f(\mathbf{x}_{n+1}). \tag{2.34}$$

After sorting the vertices, the *centroid* $\bar{\mathbf{x}}$ of the simplex can be calculated using the vertices $\mathbf{x}_1, \ldots, \mathbf{x}_n$[27], which will be used for further simplex operations. The centroid can be viewed as an average of all vertices except the worst one, and is calculated by [27]

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i. \tag{2.35}$$

An illustration of the simplex when $n = 2$, in addition to the position of the centroid can be seen in figure 2.14 (a). After the centroid has been calculated, the four operations can be described more precisely [27]:

1. **Reflection.** The reflection point $\mathbf{x}_r$ can be understood as reflecting the simplex away from the worst vertex, in hopes of finding a more optimal vertex. This new vertex is calculated using [27]

   $$\mathbf{x}_r = \bar{\mathbf{x}} + \alpha(\bar{\mathbf{x}} - \mathbf{x}_{n+1}). \tag{2.36}$$

   An illustration of this operation for the case $n = 2$ is shown in figure 2.14 (b). Which operation to perform next depends on the objective function value of the reflection point.

2. **Expansion.** The expansion point is calculated when the reflection point yields a lower function value than the previous worst simplex [27]. It can be understood as testing a point further away from the simplex in the direction of the reflection point, in order to see if this point yields an even lower function value. The expansion point is given by [27]

$$\mathbf{x}_e = \bar{\mathbf{x}} + \beta(\mathbf{x}_r - \bar{\mathbf{x}}). \tag{2.37}$$

   An illustration of this operation for the case $n = 2$ is shown in figure 2.14 (c).

3. **Contraction** There are two variations of the contraction operations, which are performed depending on the objective function value of $\mathbf{x}_r$ [27]. They are called *outside* and *inside* contraction, and described as follows [27]:

   3.1. **Outside Contraction.** If the reflection point yields a function value $f(\mathbf{x}_n) \leq f(\mathbf{x}_r) < f(\mathbf{x}_{n+1})$, the region inside the simplex formed with the reflection point is investigated for a vertex with a possible smaller function value [27]. The outside contraction point is given by

$$\mathbf{x}_{\text{oc}} = \bar{\mathbf{x}} + \gamma(\mathbf{x}_r - \bar{\mathbf{x}}). \tag{2.38}$$

   An illustration of this operation for the case $n = 2$ is shown in figure 2.14 (d).

   3.2. **Inside Contraction.** If the reflection point has a higher function value than all other vertices, the inside of the simplex is probed for a vertex with a possible lower function value [27]. The inside contraction point is given by

$$\mathbf{x}_{\text{ic}} = \bar{\mathbf{x}} - \gamma(\mathbf{x}_{n+1} - \bar{\mathbf{x}}). \tag{2.39}$$

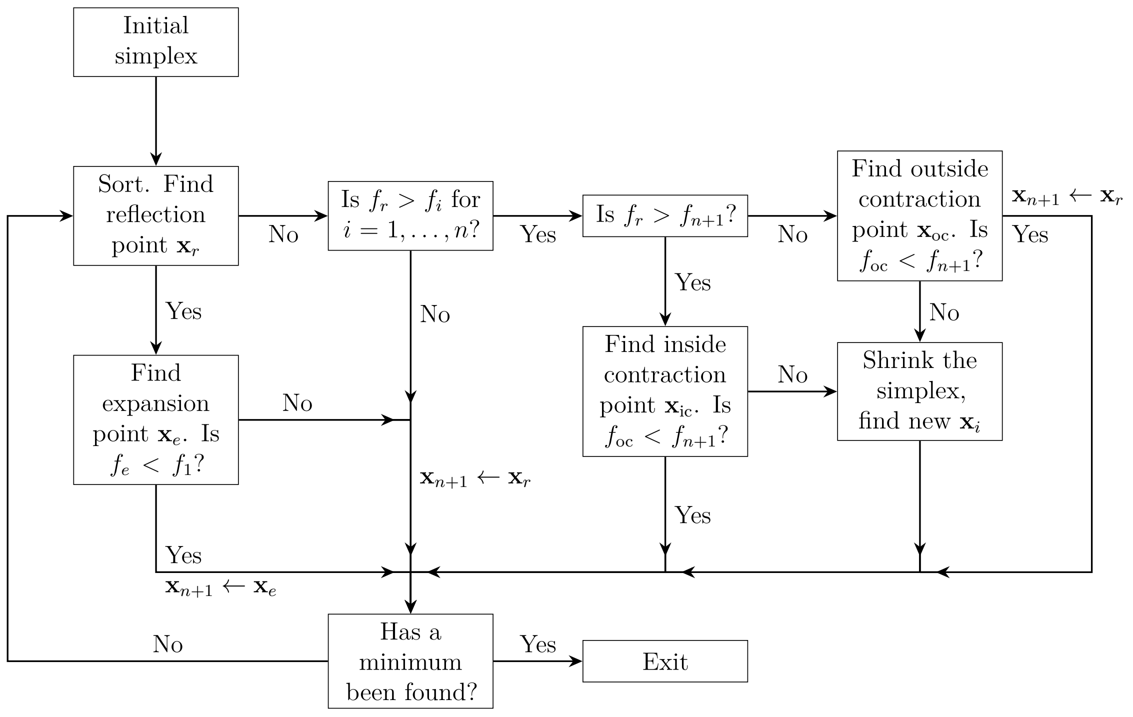
   An illustration of this operation for the case $n = 2$ is shown in figure 2.14 (e).

4. **Shrink .** If all other tested vertices yields higher function values, the algorithm assumes that the simplex surrounds the optimal vertex. To better approximate this value, the simplex is reduced in size around its best vertex [27]. The new vertices of the simplex are then given by [27]

$$\mathbf{x}_i = \mathbf{x}_1 + \delta(\mathbf{x}_i - \mathbf{x}_1) \tag{2.40}$$

   for $2 \leq i \leq n + 1$. An illustration of this operation for the case $n = 2$ is shown in figure 2.14 (f).

Using the different simplex operations the algorithm procedure is best understood by following a flowchart, such as the one shown in figure 2.15, where the shorthand notation $f_i = f(\mathbf{x}_i$ has been used for convenience.



Initial simplex
Sort. Find reflection point $\mathbf{x}_r$
No
Is $f_r > f_i$ for $i = 1, \dots, n$?
Yes
Is $f_r > f_{n+1}$?
No
Find outside contraction point $\mathbf{x}_{\text{oc}}$. Is $f_{\text{oc}} < f_{n+1}$?
$\mathbf{x}_{n+1} \leftarrow \mathbf{x}_r$
Yes
Yes
No
Yes
No
Find expansion point $\mathbf{x}_e$. Is $f_e < f_1$?
No
Find inside contraction point $\mathbf{x}_{\text{ic}}$. Is $f_{\text{oc}} < f_{n+1}$?
No
Shrink the simplex, find new $\mathbf{x}_i$
$\mathbf{x}_{n+1} \leftarrow \mathbf{x}_r$
Yes
Yes
$\mathbf{x}_{n+1} \leftarrow \mathbf{x}_e$
No
Has a minimum been found?
Yes
Exit

Figure 2.15: Flowchart of the Nelder-Mead minimization algorithm. The starting point of the algorithm is the top left box named *initial simplex*, and the algorithm ends when the minimization criterion has been reached. This flowchart is based on the one presented in [29] and developed further to reflect more modern implementations of the algorithm.

Data Fitting

Although the Nelder-Mead algorithm is an algorithm for finding the minimum of a function, it can be used to fit a given line to a set of data points given an appropriate objective function. Given a set of residuals r_i, a general objective function for data fitting can be given as

$$f(\mathbf{x}) = \|\mathbf{r}(\mathbf{x})\|_p = \left(\sum_i |r_i(\mathbf{x}_i)|^p\right)^{\frac{1}{p}}, \tag{2.41}$$

where $\|.\|_p$ is the p-norm [2]

2.3.3 Least Squares Regression

In nonlinear least squares data fitting, the objective function to be minimized can be written as [26]

$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m} r_i(\mathbf{x}) = \frac{1}{2}\mathbf{r}(\mathbf{x})^T\mathbf{r}(\mathbf{x}), \tag{2.42}$$

where $r_i(\mathbf{x})$ is the residual at datapoint i. The best curve fit is the local minimum point $\mathbf{x}_0$, which is the point where $f(\mathbf{x}_0) \leq f(\mathbf{x})$ for all x such that $\|\mathbf{x} - \mathbf{x}_0 < \epsilon$, where ϵ is some small positive number [26]. It can be shown that if $\mathbf{x}_0$ is a local minimum, then $\nabla f(\mathbf{x}_0) = 0$ and $\nabla^2 f(\mathbf{x}_0)$ is positive definite [26].

Gauss-Newton Method

Newton's method is a method for solving equations of the form [26]

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}. \tag{2.43}$$

This is done by finding the Taylor series approximation for $\mathbf{f}$ at the point $\mathbf{x_k}a$, which is given by [26]

$$\mathbf{f}(\mathbf{x}_k + \mathbf{p}) \approx \mathbf{f}(\mathbf{x}_k) + \nabla\mathbf{f}(\mathbf{x}_k)^T\mathbf{p}. \tag{2.44}$$

Equation 2.44 represents a linear approximation of the function $\mathbf{f}$. Setting this equation equal to $\mathbf{0}$ and solving for $\mathbf{p}$ yields an approximation for equation 2.43 [26]. To further improve the approximation of the value, however, the same process can be repeated by finding the Taylor series approximation for $\mathbf{f}$ at the point $\mathbf{p}$. This iterative process for finding successively better approximations $\mathbf{x}_{k+1}$ given a previous approximation $\mathbf{x}_k$ is usually written as [26]

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \nabla\mathbf{f}(\mathbf{x}_k)^{-T}\mathbf{f}(\mathbf{x}_k). \tag{2.45}$$

[2]More common names for the different p-norms are the *boxcar* norm, denoted $\|.\|_1$, the *Euclidian* denoted $\|.\|_2$, and the *max* norm, denoted $\|.\|_\infty$. The latter is the maximum value of a vector element in a vector.

Returning to the case of nonlinear least squares data fitting, Newton's method can be applied to find the approximation of the zero value of the gradient of $\mathbf{f}$. The gradient of $\mathbf{f}$ can be found to be [26]

$$\nabla \mathbf{f}(\mathbf{x}) = \nabla \mathbf{r}(\mathbf{x})\mathbf{r}(\mathbf{x}). \tag{2.46}$$

The *Gauss-Newton method* is this implementation of Newton's method combined with the approximation [26]

$$\nabla^2 \mathbf{f}(\mathbf{x}) \approx \nabla \mathbf{r}(\mathbf{x})\mathbf{r}(\mathbf{x})^T, \tag{2.47}$$

which assumes that $\mathbf{r}(\mathbf{x}) \approx \mathbf{0}$ for $\mathbf{x} \approx \mathbf{x}_0$ if $\mathbf{r}(\mathbf{x}_0) = \mathbf{0}$ [26]. This approximation assumes that the residuals are small, meaning this method will perform poorly when the initial guess of the iteration is a poor fit of the data. Given a step $\mathbf{x}_k$, the next step of the iterative process is given by $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$, where $\mathbf{p}_k$ is the solution to [26]

$$\nabla \mathbf{r}(\mathbf{x})\mathbf{r}(\mathbf{x})^T \mathbf{p}_k = -\nabla \mathbf{r}(\mathbf{x})\mathbf{r}(\mathbf{x}). \tag{2.48}$$

This original form of the Gauss-Newton method is rarely used directly to find the best fit. One method of calculation is using *trust region methods* [26].

Trust Region Method

Trust-region methods are ways of solving the problem stated in the Gauss-Newton method while guaranteeing convergence. In essence, this method creates a model of the objective function, and only trusts this model within some specified bounds of the input parameters. Using the trust region method on Newton's method, the model of the objective function is given as the second order Taylor series of $\mathbf{f}$ about the point $\mathbf{x}_k$ [26]

$$\mathbf{q}_k(\mathbf{p}) = \mathbf{f}(\mathbf{x}_k) + \nabla \mathbf{f}(\mathbf{x}_k)^T \mathbf{p} + \frac{1}{2}\mathbf{p}^T \nabla^2 \mathbf{f}(\mathbf{x}_k)\mathbf{p}, \tag{2.49}$$

where the bounds of the parameter $\mathbf{p}$ are given by

$$||\mathbf{p}||_2 \leq \Delta_k. \tag{2.50}$$

The value of Δ_k depends on how well the model $\mathbf{q}$ fits the objective function $\mathbf{f}$.

The trust region algorithm is as follows [26]

1. Initiate the algorithm with an initial guess $\mathbf{x}_0$ and some initial trust-region bound $\Delta_0 > 0$. Additionally some constants $0 < a$ and $b < 0$ need to be specified.

2. For $k = 0, 1, \dots$:

(a) If $\mathbf{x}_k$ is optimal, the algorithm is finished.

(b) Solve the constrained optimization problem

$$\min_{\mathbf{p}} \mathbf{q}_k(\mathbf{p}), \quad \text{where } \|\mathbf{p}\|_2 \leq \Delta_k. \tag{2.51}$$

The function $\mathbf{q}_k$ is the same as the one stated in equation 2.49. The solution to this problem will be termed $\mathbf{p}_k$.

(c) To indicate how well the model predicts a reduction in the objective function, the value ρ_k is introduced, which can be computed as [26]

$$\rho_k = \frac{\text{actual reduction}}{\text{model reduction}} = \frac{\mathbf{f}(\mathbf{x}_k) - \mathbf{f}(\mathbf{x}_k + \mathbf{p}_k)}{\mathbf{f}(\mathbf{x}_k) - \mathbf{q}_k(\mathbf{p}_k)}. \tag{2.52}$$

If ρ_k is small, the actual function value is smaller than the model, meaning that the trust region Δ_k is too large and needs to be reduced. If ρ_k is large, the trust region can be increased to include even further possible values.

(d) Calculate whether the step is a successful one or not. If $\rho_k < a$, the step is considered unsuccessful and the value $\mathbf{x}_k$ is kept for the next iteration. If the step is successful, the next iteration uses the value $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$

(e) Lastly, the value of Δ_k is updated according to the following rules:

$$\rho_k \leq a \rightarrow \Delta_{k+1} = \frac{1}{2}\Delta_k \tag{2.53}$$

$$a < \rho_k < b \rightarrow \Delta_{k+1} = \Delta_k \tag{2.54}$$

$$\rho_k \geq b \rightarrow \Delta_{k+1} = 2\Delta_k \tag{2.55}$$

3 | Experimental Setup

This section aims to provide an overview of the experimental setup used to acquire data from a MARC device and the software used to analyze such data. First, a brief explanation of the laboratory setup is given, where an account of the different components required for data acquisition of a MARC device is given. The flowchart in figure 3.1 gives an overview of how the components are related, although a more descriptive explanation of how each part of the setup works will be given. Table 3.1 gives an overview of which specific components were used in this work. The next part of this section will discuss the software used for data acquisition and analysis . Most of this section will focus on the implementation of different functions which have been used for the data analysis.

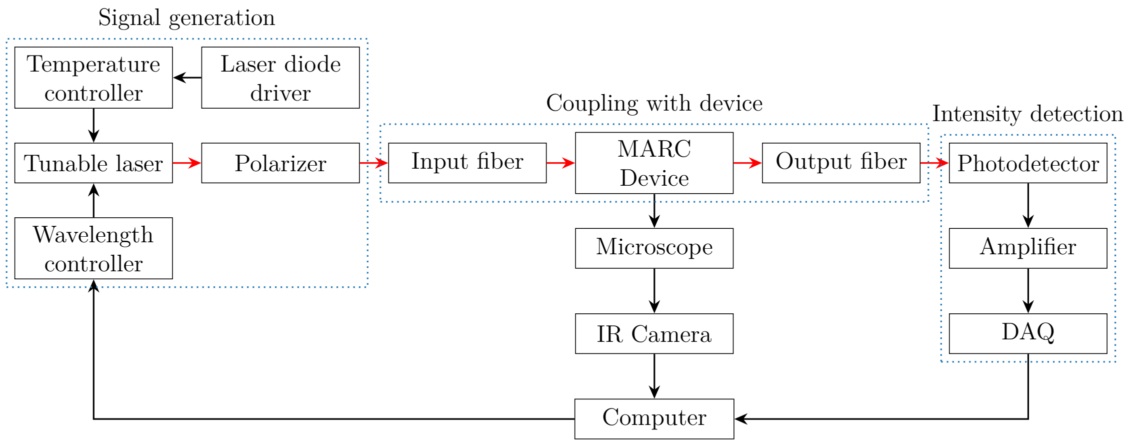


Figure 3.1: Flowchart of the experimental setup. Red arrows indicate a photonic signal, while black arrows indicate an electronic signal.

3.1 Components for Data Acquisition of MARC Device

Producing meaningful data sets from a MARC sensor requires various components. For simplicity, these components can be categorized into three essential parts, namely the *signal generation*, the *coupling* in and out of the device, and a method of *detecting* the output intensity. The data desired from a MARC device are transmission spectra showing the output intensity of the drop port of the MARC as a function of wavelength over a specified range. To achieve such a signal a *tunable laser* can be used, which can vary the wavelength of the output light by using a *wavelength controller*.

To power the laser diode in the tunable laser, a laser diode driver is required to achieve the desired current to the semiconductor diode. This current is passed through a TEC temperature controller that controls the temperature of the diode [30]. The tunable laser used in this work uses an external cavity laser in a Littmann-Metcalf configuration to tune the wavelength [31]. The laser diode is integrated into a larger resonator, which is referred to as the external cavity [32]. The beam resonates between the diode and a mirror, where it is passed through a collimating lens [32]. The Littmann-Metcalf configuration utilizes a diffraction grating within this external cavity [31, 32]. Figure 3.2 shows the different parts of the configuration. The mirror is mounted on a plate connected to a pivot point on one end and a DC servo motor on the other. The motor can retract or expand some distance δx, which adjusts the angle between the mirror and the grating. As this angle changes, a different diffraction mode will resonate within the external cavity. For a more in-depth explanation of how this works, the original article introducing the configuration can be consulted [33]. The DC servo motor has a travel distance of 12 mm and a calculated resolution of 29 nm [34], which corresponds to a wavelength shift of about 0.45 pm.

After the desired wavelength has been set, the ray exits the tunable laser in an optical fiber, which subsequently is passed into a *manual fiber polarization controller*. The polarization controller is a device that transforms the polarization of a beam propagating in a single-mode optical fiber [35]. The polarization change is achieved by utilizing stress-induced birefringence, which is caused by looping the fiber around three separate spools [35]. These three spools, called *paddles*, act as fiber retarders which respectively; change an arbitrary polarization state into a linear polarization, rotate the linear polarization, and transform the rotated linear polarization into an arbitrary elliptical polarization [35]. The three paddles can be manually rotated to achieve the desired output polarization. The polarized laser signal is subsequently focused into the input of the device waveguide by aligning the optical fiber with the facet of the input waveguide of the MARC. The optical fiber used in this work has a working distance of 14 µm and a spot diameter of 2.5 µm [14]. The light is transmitted through the device and exits at the drop port of the MARC. Next, another optical fiber is aligned at the output waveguide of the MARC device, which guides the light into a biased photodetector.

The intensity detection consists of a biased photodetector and a photodiode amplifier. The biased photodetector converts the optical signal into a current, which is subsequently amplified with a photodiode current amplifier. The amplifier then transmits the current to the *data acquisition system* (DAQ), which samples the detected current and converts the data into a digital signal sent to the computer for further processing. Another component of the setup used in this work is an optical microscope mounted atop the coupling part of the setup. The microscope is not a required component of the setup but significantly simplifies the process of aligning the input and output fibers with the photonic device. This microscope is, in turn, connected to an infrared-sensitive camera, which further helps to ensure that the fibers are aligned adequately by enabling the user to see when infrared light is transmitted through the device. A flowchart of the laboratory setup is shown in figure 3.1, while an overview of the components and model numbers of the equipment used in this work is shown in table 3.1.

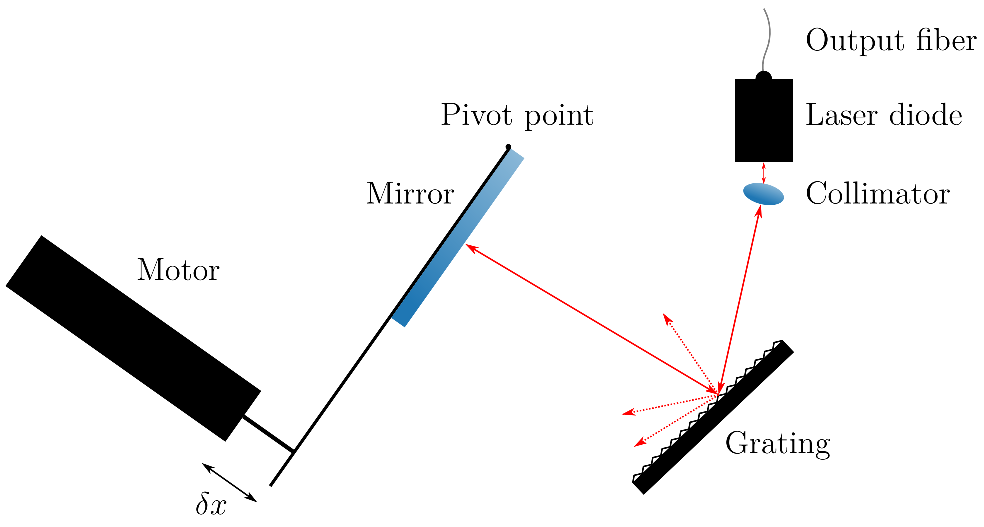


Figure 3.2: A simplified schematic of the tunable laser used in this work. The laser is an external cavity laser using the Littmann-Metcalf configuration. The distance δx illustrates the retractable distance of the servo motor, which makes the mirror adjust its angle around the pivot point marked in the schematic. The red lines indicate the laser beam moving throughout the configuration.

Table 3.1: Components and model numbers of the equipment in the experimental setup.

Component	Manufacturer	Model Number
Tunable laser	Thorlabs	TLK-L1550M
DC servo motor	Thorlabs	Z812
DAQ	National Instruments	PCI-6024E
Laser diode driver	Newport	505
Temperature controller	Newport	325
Polarizer	Thorlabs	FPC561
IR Camera	Hamamatsu	C14041-10U
Microscope	Olympus	BXFM
Photodetector	Thorlabs	DET10C2
Amplifier	Thorlabs	PDA200C

3.2 Software

All programming and data analysis in this work has been done using the Python 3.10 programming language [36]. Although most of the code which analyses the data has been written as a part of this work, some common libraries and algorithm implementations have been used. Of note are the implementations of the minimization algorithms Nelder-Mead and least squares regression, which were described in section 2.3.

3.2.1 Labview

To move the motor and record data using the experimental setup a Labview program written by Ph.D. student Jens Høvik has been used. This program works by specifying two wavelengths and subsequently drives the motor continuously from one to the other. The DAQ will continuously record data while the motor moves at a frequency of approximately 60 Hz. The user can set the motor speed, although the speed 0.01 mm/s has been used throughout this project. This continuous collection method will be referred to as the sweep method. As the two wavelengths which serve as endpoints of the sweep can be any wavelength within the range of the tunable laser, the measurement performed can be done in two different directions, depending on whether the initial wavelength is smaller or larger than the final wavelength. If the initial wavelength is smaller than the final wavelength, the direction will be referred to as increasing. If the opposite is true, the direction will be referred to as decreasing.

3.2.2 External Code Packages

External code has been used as a part of this work and will be quickly reviewed in this section. The most important code utilized is the Scipy implementations of the minimization algorithms used and the code calculating the transmission output.

Scipy Implementation of Nelder-Mead Algorithm

The Nelder-Mead implementation is the one from the Scipy package [37]. Based on the initial guess input $\mathbf{x}_0$ of size n given to the algorithm, this implementation will create an initial simplex whose dimension is $n+1$, as described in section 2.3.2. The first vertex of this initial simplex is just the initial guess $\mathbf{x}_0$ itself. The following n vertices are the initial guess vector $\mathbf{x}_0$ with variable $x_{0,i}$ multiplied by 1.05 [38] . The notation $x_{0,i}$ is used here to denote the ith vector component of $\mathbf{x}_0$, where $i = 1, \ldots, n$. The only exception is when $x_{0,i} = 0$, in which case the vector component will be exchanged with the value 0.00025 instead of being multiplied with 1.05.

The algorithm will run for a maximum of $200 \times n$ times and will stop only if the termination criteria have been reached. The termination criteria in this implementation are that the absolute difference between two consecutive simplices must be below the value `xatol` and the absolute difference between the function values of these simplices must be less than the value `fatol`. Both of these values default to 10^{-4}. The default simplex operation parameters take the same values as those given in equation 2.33.

Scipy Implementation of Least Squares Regression

The Scipy package is also utilized for its non-linear least squares implementation, which provides a wide range of possible implementations of the regression technique [39]. The default approach uses a method called *subspace trust region interior reflective* (STIR) method. This algorithm for solving least squares problems is similar to the trust region algorithm outlined in section 2.3.3. However, due to the complexity of the mathematical framework required to explain this adequately, a further elaboration on the approach is considered out of scope for this thesis. The interested reader is advised to consult the paper in which the approach was initially proposed [40]. As can be seen from equation 2.49, the Jacobian of the objective function is required for this computation. Since an analytical expression for the derivatives of the residuals is impossible to acquire, a numerical approximation is required to complete the computation. In the Scipy package, this is done by using a finite difference approximation [41].

This implementation of least squares uses the same termination criteria as the implementation of Nelder-Mead, though with different default values. These are $100 \times n$ for the number of function evaluations and 10^{-8} for `ftol` and `xtol`. The least squares implementation has an additional termination criterion which acts in the same way as `ftol` and `xtol`, although it depends on the difference in the norm of the gradient. This criterion is called `gtol` and has the same default value as `ftol` and `xtol`.

Analytical Model for MARC Devices

To simulate the transmission responses of different MARC devices a script calculating this as described in equation 2.27 was used. Mukesh Yadav originally wrote the script in Matlab, while Espen Hovland translated the script to Python during his work on his project thesis in the Fall of 2021. This script has also been used for simulations in this work, with some minor additions added. These modifications include adding the option to retrieve zero-centered intensity and the ability to input an array of refractive indices (i.e. refractive index that changes depending on wavelength) instead of only having a fixed refractive index. The entirety of the script can be found in appendix A.

3.2.3 Code Written in This Work

The code written in this work consists of three different Python files. One which reads a folder containing MARC sensor measurements and returns the resonance shift of each ring, one which contains the framework behind the file just mentioned, and lastly, a file that controls the DAQ and motor. This last file enables the user to perform measurements based on a step approach rather than a sweep approach. These three files are given in their entirety in appendices B, C and D, respectively.

Curve Fitting

The approach employed in this work to estimate the resonance shift of a ring in a MARC sensor is by finding the best curve fit to each measurement exhibiting the shift and calculating the shift based on the values of these curve fits. In order to find such a curve fit, the equation calculating the MARC intensity output given in equation 2.27 is used together with the minimization algorithms discussed in section 2.3. For the minimization algorithms to work correctly, a residual function has to be defined, which accepts a vector $\mathbf{x}$ of parameters to be optimized by the algorithms.

The parameters to be improved must be chosen so that the minimization algorithms can adequately fit the curve to the data. As seen in the discussion in section 2.2.2, several parameters are determining this signature. The number of parameters depends on the number of rings in the sensor and the refractive index approximation used.

The parameters chosen to be improved upon are the ring radius r_i, angular separation θ_i and horizontal shift $\delta\lambda_i$ of each ring $i \in \{1, 2, \dots\}$. Additionally, an assumption is made that each ring has the same coupler coefficient t_1 and loss coefficient α, which are treated as parameters to be improved upon as well. Lastly, the refractive index n_{eff} is used as a parameter. However, as different approximations for the refractive index were tested throughout this work, this last parameter of the vector $\mathbf{x}$ has a variable length as well. This is because the different n_{eff} models require a different number of paramters. The vector $\mathbf{x}$ can then be expressed mathematically as

$$\mathbf{x} = [r_1, \theta_1, \delta\lambda_1, \dots, r_n, \theta_n, \delta\lambda_n, t, \alpha, n_1, \dots, n_m]. \tag{3.1}$$

where n is the number of rings in the device, and m depends on the approximation of the refractive index. If a constant approximation is chosen, $m = 1$ and $n_1 = n_{\text{eff}}$. If the Sellmeier approximation is selected, m varies with the number of Sellmeier coefficients chosen, but is in this program either $2, 4$ or 6. In this case the elements in the vector are $n_1 = B_1, n_2 = C_1, n_3 = B_2, n_4 = C_2, \dots$.

This vector $\mathbf{x}$ is passed to the minimization algorithms. The minimization algorithms then use the analytical model for the intensity, provided in equation 2.27, to find the best curve fit for the data. The minimization algorithms will return the parameters in the same form as is shown in equation 3.1 which gives the optimized curve fit.

Calculating Resonance Shift

The complete process of calculating the resonance shift of a MARC device is shown in figure 3.3. The process starts with collecting transmission measurements exhibiting a resonance shift, where each data set has resonance peaks at different wavelengths. The transmission measurements are then preprocessed, where unwanted data points are removed. Such unwanted data points include duplicate measurements at the start and end of each file, resulting from how the Labview program saves the measurements. The data points are then sorted such that wavelengths and intensities are in the correct order, independent of the sweep direction. The final step of the preprocessing is normalizing the data between -1 and 1.

Next, the *a priori* information known about the MARC needs to be entered. This information is the radius of each ring in the device and their angular separation. Additionally, information concerning the effective refractive index needs to be entered here, regardless of whether the constant or the Sellmeier approximation is used. Next, a horizontal wavelength shift of the whole signature must be found to achieve a better starting point for the minimization algorithms. This shift could be due to frictional hysteresis or calibration inaccuracy of the laser, but it is not known for sure. However, an estimation of such a shift is required to achieve a good fit with each resonance peak. Lastly, an iteration is performed where the Sellmeier coefficients are improved to fit the data better.

After the preceding steps have been performed on the first transmission measurement of the data set, the final curve fit can be made for the data, yielding optimized parameters. The optimized parameters for this measurement can then be used as an initial guess for the subsequent measurement in the data set. An assumption is then made that none of the parameters will change between measurements, except the resonance shift. Finally, a curve fit is found for all transmission measurements in the data set by restricting the bounds on all parameters except the resonance shift. The difference in peak positions is recorded for each iteration of curve fitting. When the first such shift has been recorded, this can be used to estimate the next resonance shift, providing the subsequent curve fit an improved initial guess. When a curve fit has been made to all transmission measurements, the program exits and yields the calculated resonance shifts.

Motor Control

A Python program was written as part of this thesis to move the motor a set distance before taking a measurement and moving on further. This approach ensures that each data point has a set wavelength, making repeated measurements possible

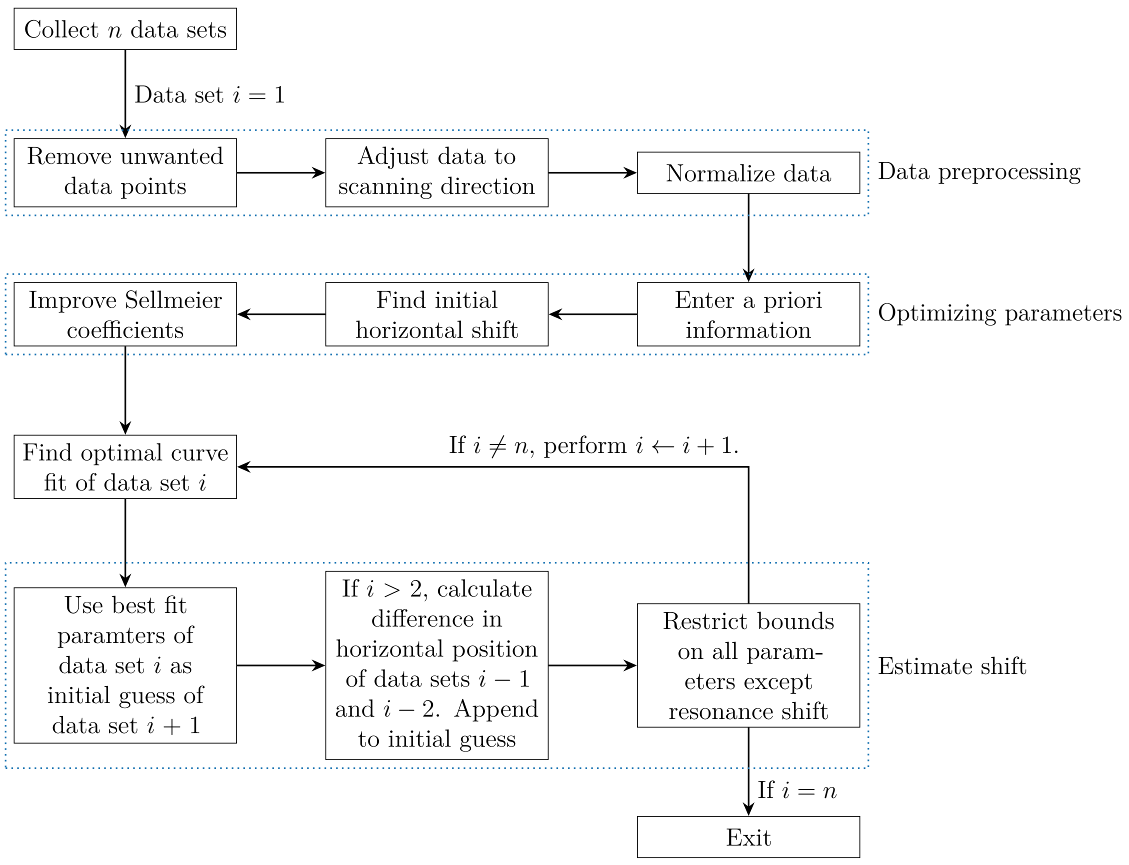

Collect n data sets
Data set $i = 1$
Remove unwanted data points
Adjust data to scanning direction
Normalize data
Data preprocessing
Enter a priori information
Find initial horizontal shift
Improve Sellmeier coefficients
Optimizing parameters
Find optimal curve fit of data set i
If $i \neq n$, perform $i \leftarrow i + 1$.
Use best fit paramters of data set i as initial guess of data set $i + 1$
If $i > 2$, calculate difference in horizontal position of data sets $i - 1$ and $i - 2$. Append to initial guess
Restrict bounds on all parameters except resonance shift
Estimate shift
If $i = n$
Exit

Figure 3.3: Flowchart showing the procedure to calculate a resonance shift across a collection of n data sets.

where the intensity is measured at the same wavelength each time. This method of acquiring data will be referred to as the "step" method. The code of this program is shown in appendix D. To communicate with the DAQ the API provided by National Instruments was used [42]. For interaction with the servo motor, the API provided by Thorlabs was used [43].

4 | Methods

This chapter aims to describe the methods and procedures employed in this work. These are divided into three sections, the contents of which are described as

1. Using the experimental setup described in section 3.1 in order to acquire transmission spectra for MARC sensors.
2. Investigating the limitations of the experimental setup, where special attention has been paid to the sampling techniques.
3. Improving the procedures relating to the data analysis of the transmission spectra.

4.1 Data Acquisition

In this section, the procedures for performing measurements will be described. Each step required to acquire transmission spectra are listed as follows:

1. Powering on all the equipment described in 3.1, except the output power of the laser driver.
2. Placing the device on the sample holder, and using the microscope to approximately align the input and output fibres to the waveguide counterparts.
3. The laser driver output is then turned on, with an output current of 396 mA. Positioning the IR camera at the output ports of the device, the position of the input fibre is carefully adjusted until light is seen exiting the ports, as shown in figure 4.1a. This light is due to scattering from the waveguide, and may not always be as visible depending on the waveguide. Figure 4.1b shows an example of a properly aligned input optical fiber.
4. Next, the position of the output fibre is adjusted until a maximum signal strength is observed on the amplifier display.
5. To further maximize the signal strength, the polarization of the input light can be adjusted by changing the angles of the paddles on the manual fiber

polarization controller.

6. When the maximum signal strength has been found, measurements can be taken using either the Labview program or the python program, as explained in section 3.2.

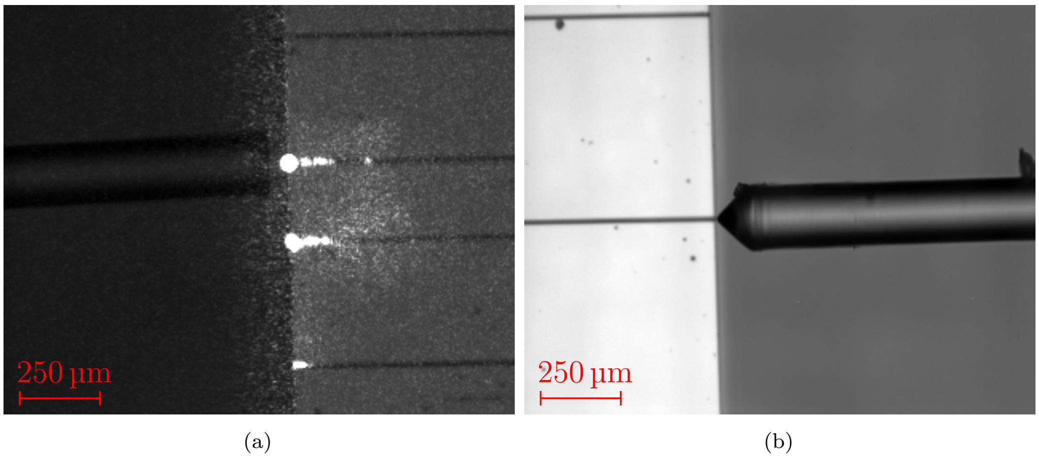


Figure 4.1: Coupling of lensed fiber and waveguide structure for (a) the output coupling and (b) the input coupling. Both images are taken with the camera described in table 3.1, using an IR lens. This makes it possible to observe the IR light, which can be seen at the output coupling as bright spots as the light scatters from the waveguide.

4.2 Investigation of the Limitations of the Experimental Setup

The laboratory setup shown described by the flowchart in figure 3.1 has been used previously for transmission measurements of waveguides by the research group. In this thesis, the workings of the setup have been investigated to identify weaknesses in the acquisition procedures, and to implement improvements. In particular, issues concerning the spacing of data points and the performance of the DC servo motor have been of special interest.

First, the different noise contributions in the setup need to be outlined. The different noise sources considered were the detector, coupling losses, losses due to scattering in the waveguide, and inconsistencies in the servo motor.

4.2.1 Hysteresis

Signs of hysteresis were observed when performing transmission measurements as a function of wavelength. As described in section 3.2.1, the transmission measurement can be performed by sweeping the wavelength in an increasing or a decreasing manner. The observed signs are that the resonating peaks of the transmission spectra appeared at different wavelengths depending on the sweep direction. This causes an inaccuracy in determining the position of the resonating peaks, in addition to reducing the reproducibility of the transmission measurements acquired using the setup.

Frictional hysteresis can be observed in servo motors, where the hysteresis can be classified into two different regimes: pre-sliding friction and gross-sliding friction [44, 45]. The pre-sliding friction is primarily dependent on the displacement before sliding, while the gross-sliding friction is due to the sliding velocity [44, 45].

It will be investigated if the system suffers from hysteresis, and if so, whether this hysteresis is position- or velocity dependent. To determine whether the system suffers from hysteresis transmission measurements will be performed while sweeping in the forwards and backwards directions using the Labview program. If hysteresis is present in the actuator, a distinct discrepancy will be observed in the position of the resonating peaks. To determine whether the frictional hysteresis is due to position or velocity, the same transmission measurement will be performed using the step method. The step method will alleviate the hysteresis if the friction is caused by the velocity, as the method stops completely before performing each measurement.

4.2.2 Spacing of Data Points

When performing transmission measurements as a function of wavelength on a waveguide device using the Labiew program described in section 3.2.1, the data points procured are not equally spaced. To gain a better understanding of the cause of this effect the specific wavelengths sampled are investigated to identify any inconsistencies between separate measurements. The consequences of the effect are mainly that repeated measurements under the same conditions yield data sets with a different number of samples. This is due to measurements are taken between a set range of wavelengths, and when the spacing between two sampled wavelengths vary between each measurement, the number of samples within a set range will differ. There are two possible explanations for this discrepancy in the spacing of data which will be explored. These possible causes for unequal spacing are due to the acceleration of the motor when doing a sweep measurement, and the synchronization of the DAQ and the wavelength controller. A way to mediate the unequal spacing of data is to use the step method, which will alleviate both these problems. However, due to the step method using a very long time to perform measurements involving many data points, it is useful to consider other remedies for the unequal spacing as well. One such option is to interpolate the data acquired using the Labview program.

To investigate the problems concerning the spacing of data points transmission measurements will be performed using both the Labview program and the step method. The mean and standard deviation of these measurements will be calculated and compared againt one another. Additionally, an interpolation will be performed using the `interp` function, which is built in with the Numpy library [46] The mean and standard deviation of the interpolated measurement will also be calculated and compared with the corresponding values of the two data acquisition techniques. The peak position of a resonance peak will be compared before and after interpolation, to investigate whether the interpolation will affect this.

4.3 Development of Data Processing Procedures

After retrieving data sets from the experimental setup, the measurements will be analysed. The goal of this analysis is to develop a program which automatically detects a resonance shift caused by a change in the refractive index of the media surrounding the ring resonator of the MARC device. To arrive at this final product, there are several considerations concerning the data processing which have to be taken into account.

4.3.1 Normalization

The first consideration to take is that the transmission measurements performed over a specified wavelength range have different intensity values depending on how much light is transmitted through the waveguide. This results in different measurements having a varying peak intensity value, making both comparisons and data fitting more difficult. A solution to this issue is normalizing the data between 0 and 1, which essentially makes the intensity units arbitrary when examining the plots. However, as the interesting property of these plots are the given signature, and not the specific intensity value this is of no issue. Thus, such normalization will be applied to all measurements. Such a normalization can easily be obtained by dividing every intensity measurement by the maximum intensity value obtained.

Another normalization strategy is zero-centering the data and restricting it between the range -1 and 1. Zero-centering is performed by subtracting every data point by the mean of the data set. This shifts the plot down vertically, so that the mean of the data set is 0. The reason for investigating this normalization is that it has shown improved results for other types of data sets [47–49]. Both these normalizaion strategies will be evaluated.

4.3.2 Data Fitting Using *a priori* Knowledge

An analytical model of the MARC sensor can be calculated using equation 2.27. By inserting a priori knowledge such as the ring radius r and angular separation θ of the measured device, the model should theoretically compare nicely with the recorded data. In practice, however, this is not the case. Only inserting this a priori knowledge does not compare well with the collected data.To be able to match the analytical model to the data, a curve fit can be performed. However, since the analytical model is calculated by a non-linear function, some more advanced techniques are required compared to a normal linear fit.

As there exists a plethora of different non-linear minimization algorithms, a wide selection of such algorithms could be investigated in order to find one which is optimal for this category of data sets. However, to fit the scope of this work only two algorithms were investigated, namely the Nelder-Mead algorithm and the non-linear least squares approach. These were chosen to see if there is any signaificant difference in using a mathematical and a heuristic minimization approach, as explained in section 2.3. Methods for further improving the initial guess in addition to using the a priori information is investigated. This helps to improve the performance of the minimization algorithms.

4.3.3 Comparison of Different Norms in Objective Function

Although the least squares approach only works on a 2-norm objective function, the Nelder-Mead algorithm can use any function as its objective function. As the general p-norm of the residuals is chosen, as explained in section 2.3.2, different values for p can be selected to see how they affect the curve fitting. The norms which will be tested are the boxcar, Euclidian and max norms, i.e. $p = 1, 2$ and ∞.

4.3.4 Comparison of Refractive Index Models

An approximation used when calculating the analytical transmission response of a MARC device is to ignore dispersion, i.e. using a constant refractive index across the different wavelengths of the laser. An investigation will be done to see whether better approximations to the data can be achieved by using a more accurate model of the dispersion, such as the Sellmeier equation, which is described in section 2.1.4.

4.3.5 Residual Analysis for Noise Characterization

As the objective functions used for curve fitting are determined by the residuals of the measurements, it is of interest to investigate the resiudals after a good curve fit has been found. This will indicate whether the curve fit is a good one, and will describe the noise present in the measurements. It is assumed that the noise present in the measurements is white noise, i.e. normally distributed noise. The least squares approach assumes this. It is good to have an idea of what to expect from a data set containing normally distributed noise. This can be achieved by using the analytical mode to produce an ideal, noise-free intensity curve, which white noise can be added to. By adding white noise with $\mu = 0$ and $\sigma = 0.05$, and comparing this with a curve fit of the same analytical model, yields the residuals shown in figure 4.2. These values for the noise are chosen as they produce noise of similar size as those found in the experimental data.

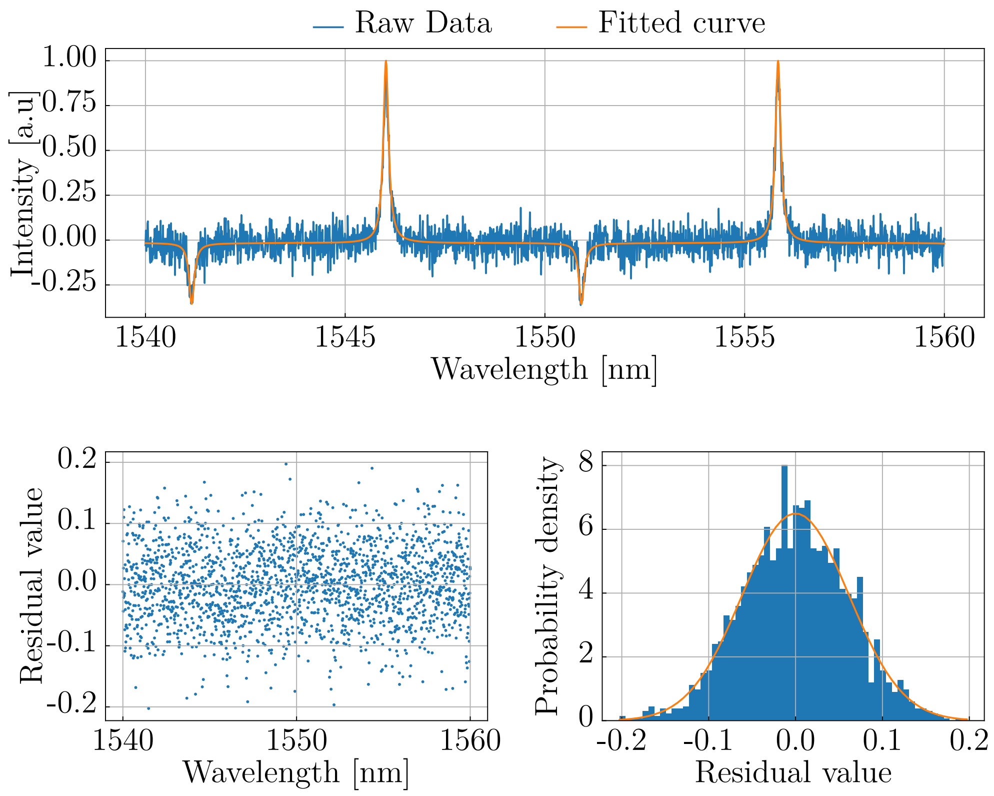


Figure 4.2: Top image shows a good curve fit with low cost-value. The bottom left image shows the residuals plotted against wavelength. The bottom right image shows the probability density against residual value, with a best fit Gaussian curve.

4.3.6 Validity of Model

To test the validity of the preceding considerations, the program developed will be performed on a set of transmission measurements exhibiting a resonance shift. The calculated resonance shift will be compared against the calculations made in the article *Multiplexed Mach-Zehnder interferometer assisted ring resonator sensor* [8] by Mukesh Yadav. The goal is to compare the resonance shift calculated by the algorithm devised in this work with the resonance shift calculated by Yadav in his article.

As stated in the article, solutions of different saline concentrations were pumped through a microfluidic channel covering the ring resonators of the MARC sensor. The pump rate at which the saline flowed through the channels was 20 µL/min [8]. As the concentrations of the saline increases, a resonance shift of the MARC signature is expected. The results in the article which will be compared are the resonance shifts observed for a MARC device with a single ring of angular separataion 135°, corresponding to sample 2 in table 5.1 and a resonance shift observed in a MARC with two rings of angular separations 135° and 180 degree, corresponding to sample 3. In the experiment using sample 2, the only ring present in the MARC was exposed to the different saline concentrations. In the experiment using sample 3 only the 135° ring was exposed to different saline concentrations, while the 180° ring was continually exposed to water.

Both experiments performed resulted in four different data sets, one for each concentration of the saline. The different concentrations investigated were 0 %, 6 %, 12 % and 18 %. The results found by Yadav are repeated in table 4.1

Table 4.1: Paramters for resonance shift obtained in article by Yadav [8]

Ring (180°)	Ring (135°)	$\Delta\lambda_{180°}$ [nm]	$\Delta\lambda_{180°,\text{eff}}$[nm]	$\Delta\lambda_{135°}$[nm]	$\Delta\lambda_{135°,\text{eff}}$[nm]
Water	Water	0	0	0	0
Water	6% Saline	~0.02	~0.02	0.52	0.52
Water	12% Saline	0	~0.02	0.47	0.99
Water	18% Saline	0	~0.02	0.50	1.49

The goal of the comparison is to see whether the calculated resonance shifts are the same, and if not, how much they differ. As the resonance shift of the 180° ring is only given approximately, some differences are expected to be found. However, if the calculations are done correctly, the calculated shifts should be approximately the same.

5 | Results and Discussion

The results will follow the outline described in section 4, namely first investigating possible limitations of the experimental setup before reviewing methods of processing the data sets obtained from the experimental setup. Lastly, the validity of the algorithm described in figure 3.3 will be determined. As there are different areas of interest in the results, it has been considered more useful to present the results combined with the discussion. This makes it easier to relate the different parts of the discussion to the corresponding parts of the results.

In this section, comparisons of minimization algorithms will be considered, in addition to comparisons of variations of the same minimization algorithm. When such comparisons are made, the term *cost value* will be used, which is the value of the objective function of the minimization algorithm after the optimized parameter has been found. When comparing cost values, i.e. the return value of the objective function of a minimization algorithm, some considerations have to be taken. For the Nelder-Mead algorithm, the cost value will be the value returned of the objective function, given in equation 2.41. However, due to the way the least squares algorithm is implemented, the value given by equation 2.42 cannot be directly compared to the Nelder-Mead cost value. In order to compare these, the cost value of the least squares algorithm will be given as $\sqrt{2f(\mathbf{x})}$, where f is given by 2.42. By doing this, the cost values of the two objective functions will be the same if the p-norm used in equation 2.41 is the 2-norm, i.e. the Euclidian norm.

Three different MARC sensors will be considered when presenting the results, which will be referred to as samples $1, 2$ and 3. Their specific characteristics are outlined in table 5.1. The data from sample 1 is gathered as a part of this project using the setup and procedure explained previously. The transmission measurements of the following two samples are performed by Mukesh Yadav using the same experimental setup. On sample 3, a sensor measurement was performed, resulting in a resonance shift of the MARC signature. The measurements exhibiting this shift consisted of four data sets and were used by Yadav in his article *Multiplexed Mach-Zehnder interferometer assisted ring resonator sensor* [8].

Table 5.1: The different MARC samples used for data collection. The sample number is arbitrary, and serves only as a convenient way of referencing the individual samples.

Sample no.	No. of rings	Ring radius [µm]	Ang. sep. [°]
1	1	20	180
2	1	45	135
3	2	25	180
		45	135

5.1 Concerning the Experimental Setup

Different aspects of the experimental setup will be considered. As the experimental setup comprises many components, there is room for error from several sources. First, some different sources of noise contributions will be considered. Next, the issues concerning actuator hysteresis will be considered before ending the section with a discussion regarding the spacing of data points.

5.1.1 Noise Contributions

In the detection and recording part of the setup, three components are the main sources of noise and inaccuracies in the measurements taken. In particular, the components are the InGaAs detector, the amplifier, and the DAQ.

DAQ Precision

The data detected by the photodetector is recorded by transmitting the detected light to the amplifier, which subsequently sends the signal to the DAQ. The DAQ, in turn, sends the signal to the computer, as was explained in section 3.1. According to the user manual of the amplifier, it will deliver a DC voltage proportional to the display reading of the photodiode current [50].This linear relationship is between the voltage range [0 V,10 V] and the *display reading* range [0,10000]. The display reading depends on which of the current display options are set, where the options are nA,µA and mA. Additionally, for each order of magnitude of the unit, there is different options as to how accurate the measured current is. For nA there are two options, namely `000.00` and `0000.0`. For µA there are three options, which are `00.000`,`000.00` and `0000.0`, while for mA there is only one option, which is `00.000`. This means a detected current of 20 nA, which can be observed as `020.00` in the display, will be received by the DAQ as a voltage current of 2 V. While the same detected current with the other display option for nA will be observed as `0020.0` in the display and thus be read by the DAQ as a voltage of 0.2 V

The DAQ manual reports a precision of 4.88 mV for the input range [−10 V, 10 V] [51]. The precision of the measured current thus depends on the display option chosen on the amplifier. A current read as `020.00`nA will thus have a precision of 0.0488 nA, while the same current read as `00.020`µA will have a precision of 0.0488 µA

Amplifier Noise

The noise caused by the amplifier and detector were investigated. First, a measurement was taken with only the amplifier and DAQ turned on before a second measurement was taken with the detector turned on as well. This was done to determine the noise caused by the amplifier and detector separately. The data of the two measurements can be seen in figure 5.1, while the mean μ and standard deviation σ of the measurements are shown in table 5.2.

Table 5.2: Mean and standard deviation of signal detected in amplifier, measured separately with the detector either turned on or off.

Detector state	Mean [nA]	Standard deviation [nA]
Off	0.00114	0.00653
On	0.829	0.0189

5.1.2 Hysteresis

Transmission measurements were performed to determine if frictional hysteresis is present in the servo motor. As shown in figure 5.2, there is a clear indication of hysteresis. This is seen as the resonance peaks differ in their position depending on the direction of the wavelength sweep. An indication as to why this is a product of hysteresis and not some other factor causing resonance shifts is that the resonance shift is only present at some of the resonance peaks. This is further corroborated by the fact that the step method yields similar hysteresis, as shown in figure 5.3. The fact that the step method yields similar hysteresis as the data procured using the Labview program shows that the hysteresis is position-dependent and not velocity-dependent.

Although compensation techniques for hysteresis can be found in the literature [52, 53], such methods have not been explored in this thesis due to time constraints. However, when performing repeated measurements, the direction of the sweep should be consistent to avoid unexpected shifts of resonance peaks. Additionally, it is advised to lubricate the rotational element of the servo motor [34], which can be further investigated to see whether this affects the hysteresis.

5.1.3 Spacing of Data Points

To mitigate the discrepancy in step lengths between data points, three alternatives are investigated, namely the sweep method, an interpolation of the sweep method

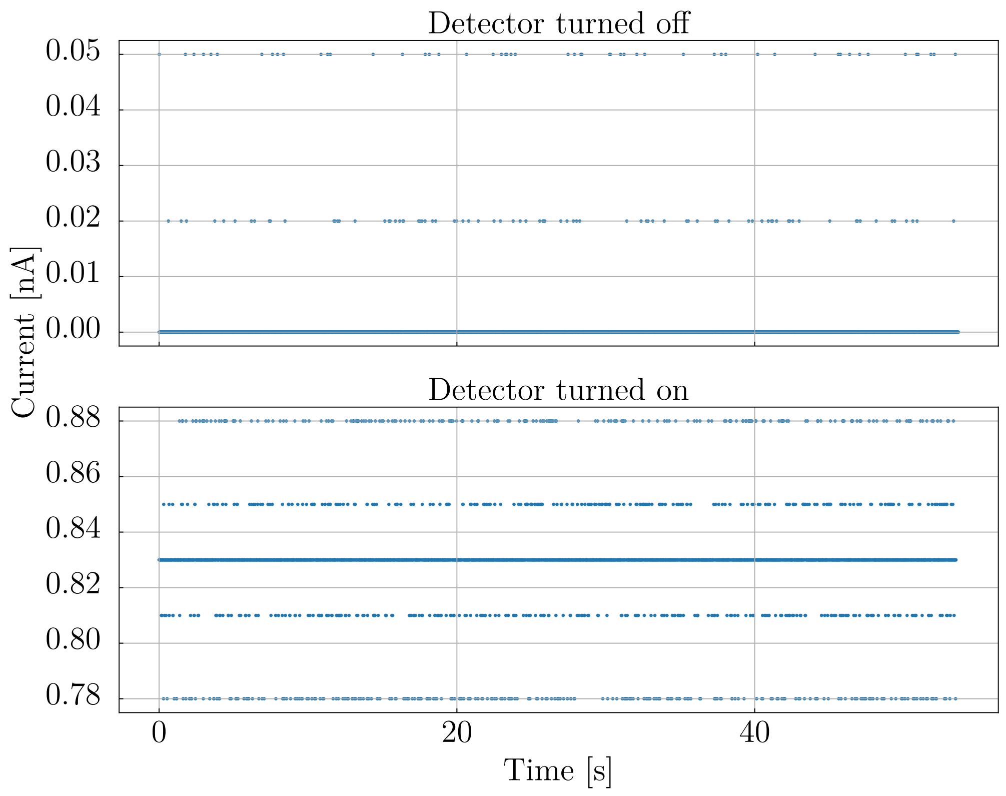


Figure 5.1: The detected current by the amplifier with no input optical signal. The top image shows the current detected with the photodetector turned off, while the bottom image shows the current detected with the photodetector turned on.

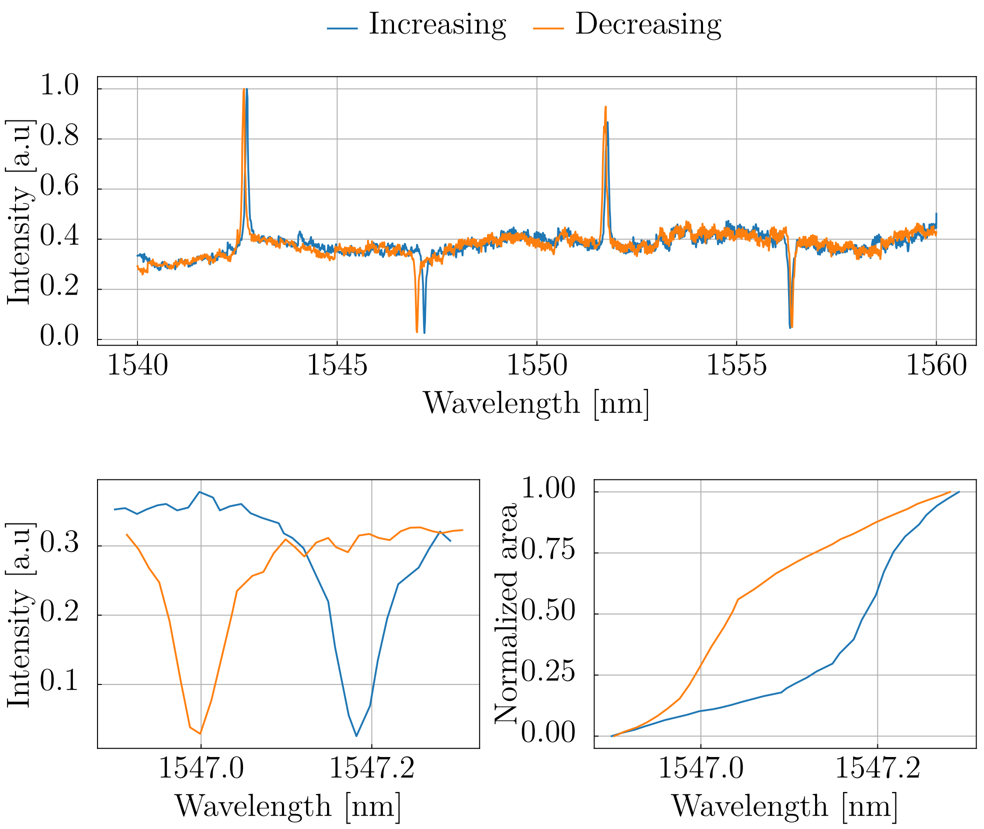


Figure 5.2: Hysteresis in a transmission measurement captured with the Labview program. The hysteresis can be seen when comparing an increasing wavelength sweep to a decreasing wavelength sweep. The top image shows the transmission data for measurements performed in an increasing and a decreasing direction. The bottom left image shows a magnified view of the hysteresis observed in the top image. The bottom right image shows the normalized area between the curve and a horizontal line at 0.4 intensity.

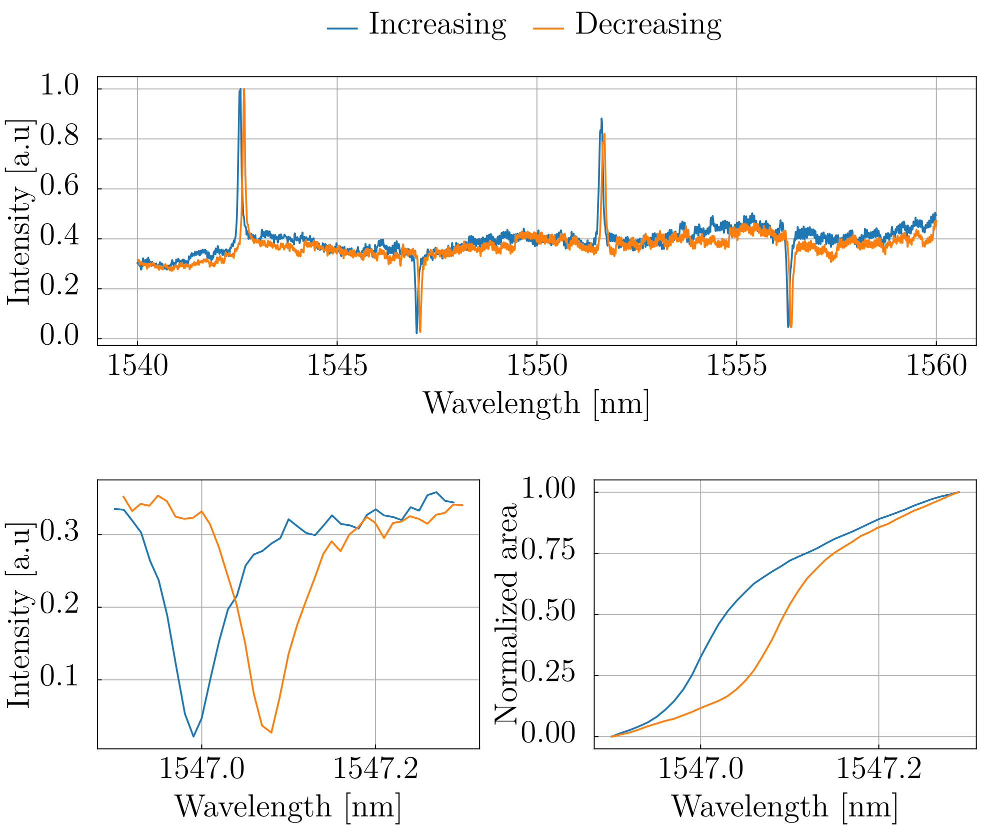


Figure 5.3: Hysteresis in a transmission measurement captured using the step method. The hysteresis can be seen when comparing an increasing wavelength sweep to a decreasing wavelength sweep. The top image shows the transmission data for measurements performed in an increasing and a decreasing direction. The bottom left image shows a magnified view of the hysteresis observed in the top image. The bottom right image shows the normalized area between the curve and a horizontal line at 0.4 intensity.

and the step method. The mean and standard deviation of the spacing between data points for the three methods are shown in table 5.3.

The sweep method shows a significant standard deviation in its gathered data, with the standard deviation being approx. 25 % of the mean value. By comparison, the standard deviation of the interpolated data is approximately zero, while the step method has a standard deviation which is 3.8 % of its mean value.

Table 5.3: Mean and standard deviation of spacing of points

Method	Mean spacing [nm]	Standard deviation [nm]
Sweep	0.0125	0.00317
Interpolation	0.0100	9.29×10^{-14}
Step	0.0100	0.000

Although the interpolation yields a very low standard deviation of the data spacing, other considerations need to be taken into account. When doing sensing experiments, the exact position of the peaks is essential. Figure 5.4 shows a zoomed-in image of such a peak, with both the raw data and the interpolated line. As can be seen in the figure, the peaks are very close, but a slight difference occurs during the interpolation. The peak of the raw data is at wavelength 1542.666 nm, while the interpolated peak is at 1542.679 nm, meaning the interpolation results in a 13 pm shift. Considering that the exact position of the resonance peak is of interest, this shift is preferably avoided.

5.2 Optimization of Curve Fitting

In order to arrive at the algorithm presented in figure 3.3, several aspects of data processing need to be considered, as discussed in section 4.3. To summarize, the main aspects to consider are the normalization, the norm used in the objective function, and the refractive index approximation. The results of these considerations will be described in this section.

5.2.1 Curve Fitting Using Few Variables

As a naive first approach, one can use the analytical model as stated in equation 2.27 to curve fit the data. The minimization algorithms can be used to find a curve fit using only the knowledge of the ring radius and the angular separation of the drop port and throughput port of the ring resonator in the MARC. Some values necessary to perform the calculation are $t_1 = 0.95$, $\alpha = 0.98$ and $n_{\text{eff}} = 3.9$, which are some approximate values for a non-ideal waveguide. The resulting curves by the minimization algorithms performed on sample 1 are shown in figure 5.5. As seen here, neither of the curves fit very well with the data.

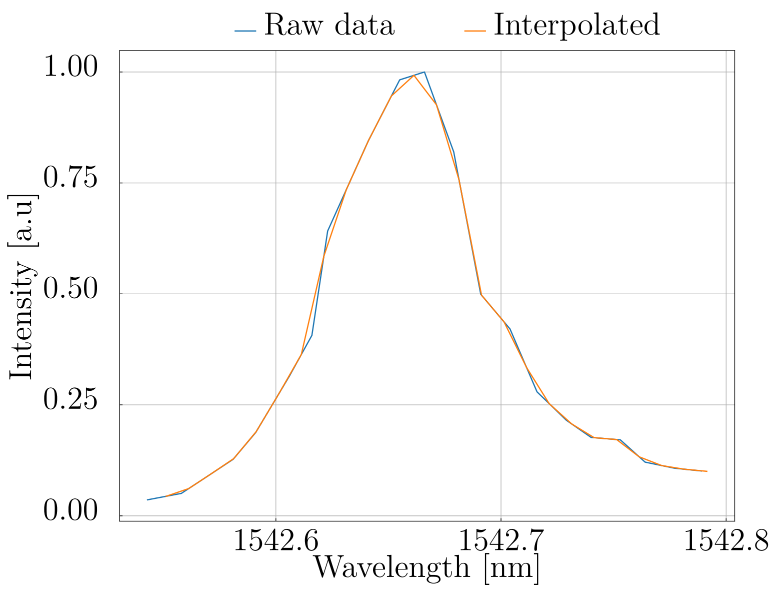


Figure 5.4: A magnified view of a resonance peak in a transmisison response. The data was acquired using the Labview program, which has data points of unequal spacing. The interpolated data has equally spaced data points, but slightly shifts the resonance peak value.

The Nelder-Mead algorithm fits some resonance peaks well, though the overall fit of the curve is not good. This can be seen as the rest of the resonance peaks of the calculated curve do not align with the transmission measurement. As explained in section 2.2.2, several parameters may cause this. The ones of note for the separation between peaks to fit well are the ring radius and the effective refractive index.

For the least squares algorithm, the curve does not appear to fit well. This is due to the algorithm finding some local minimum which minimizes the cost function, but which is not a good fit overall for the function. Although small residuals are desired, the noisy data will yield very fluctuating residuals. This causes there to be many local minima for the minimization algorithms to yield as a final answer. To ensure that the local minimum which the algorithms finalize on is the best curve fit, a better starting point is needed.

5.2.2 Improvement of Initial Guess

In order to provide the minimization algorithms with a better starting point, an initial horizontal wavelength shift can be estimated. A simple way to find this is to test a range of different values and choose the one with the lowest cost value. By doing this and subsequently performing the two different minimization algorithms, the curves shown in figure 5.6 are produced. This is a vast improvement over the ones produced in figure 5.5. It should be noted that this horizontal shift is not due to

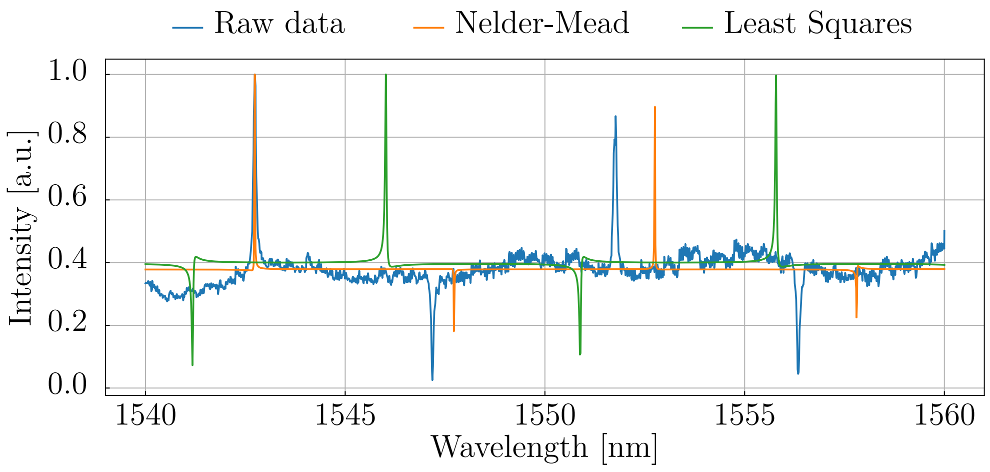


Figure 5.5: Curve fitting performed using both minimization algorithms on sample 1, with only the ring radius and the angular separation of the drop port and throughput port of the ring as initial guess.

a resonance shift but rather that the resonance peaks appear at a different position than expected by the transmission calculations. The exact reason for the shift is unknown but can be attributed to errors occurring due to the hysteresis found in the actuator or poor calibration of the laser.

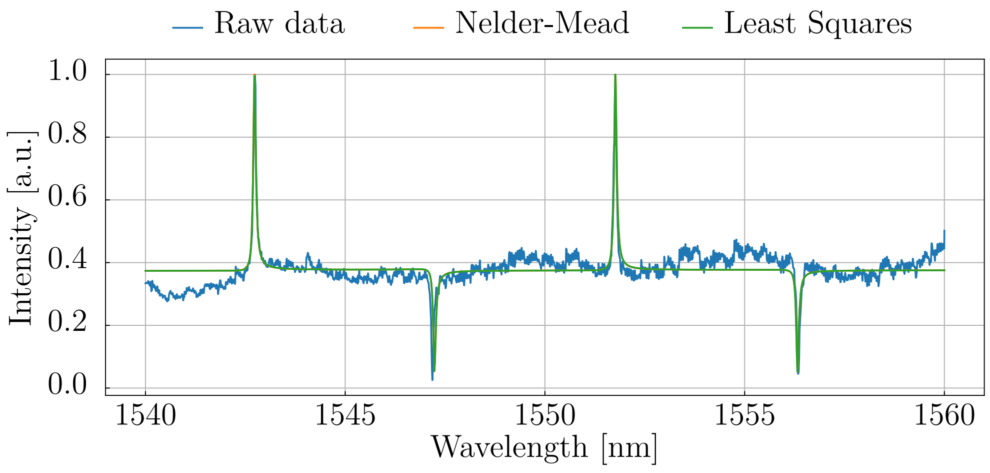


Figure 5.6: Curve fitting performed using both minimization algorithms on sample 1, with a horizontal wavelength shift estimated to provide the minimization algorithms with a better starting point. The curve produced by the Nelder-Mead algorithm is barely shown, due to the minimization algorithms yielding similar curves.

5.2.3 Zero-Centering

A simple comparison testing whether zero-centering of the transmission measurement data is beneficial is shown in figure 5.7, with the corresponding cost values shown in table 5.4. Although both normalizations yield similar curve fits, the cost values found using zero-centering are generally higher.

Table 5.4: Cost values of the different minimization algorithms with and without zero-centering the data.

Minimization algorithm	ZC Cost value	Non-ZC Cost value
Nelder-Mead	2.602	1.620
Least squares	2.602	1.619

5.2.4 Comparison of p-norms in Objective Function

Performing the Nelder-Mead algorithm with the three different p-norms $1, 2$ and ∞ on sample 1 yields the best-fit curves shown in figure 5.8. On this data set, very little difference can be found between the different norms. This might be because the analytical model fits very well with the data so that the different norms will arrive at similar optimal parameters. The same procedure was tested on sample 2 to investigate this further. The curves produced by the different algorithm variations are shown in figure 5.9. Here, a more significant difference is seen between the different approaches. The norm with the most significant deviation from the others is the ∞-norm, which has very large residuals across the whole function. Considering that the ∞-norm only seeks to minimize the maximum residual value, having a set of residuals all being close to this maximum value is possible. The 1- and 2-norm operates on a sum of all residual values, allowing some outlier large residual values as long as the majority are small.

5.2.5 Comparison of Refractive Index Approximations

The Sellmeier approximation of the wavelength-dependent refractive index was compared against a constant approximation. The amorphous phase of silicon has been intensively studied, and it is well known that the electronic and optical properties of the films are strongly influenced by deposition technique and conditions [54]. The Sellmeier coefficients of the waveguide used in this work are therefore unknown, and so an estimation of these is needed. This was done using Sellmeier coefficients for amorphous silicon as a starting point, as the waveguide in the MARC device is fabricated using this material. Amorphous silicon is a widely used material, and its Sellmeier coefficients can be found in the literature [55, 56]. However, the silicon used for the fabrication of the MARC devices does not yield the same effective refractive index as the one in the literature. Therefore, their value was estimated by iteratively using the minimization algorithms. The resulting cost values of the curves fitted with these parameters are shown in table 5.5. As can be seen in the table, the cost values are not significantly affected by this and are only reduced in the case when six coefficients are used. It is thought, however, that the curve fits can attain a lower

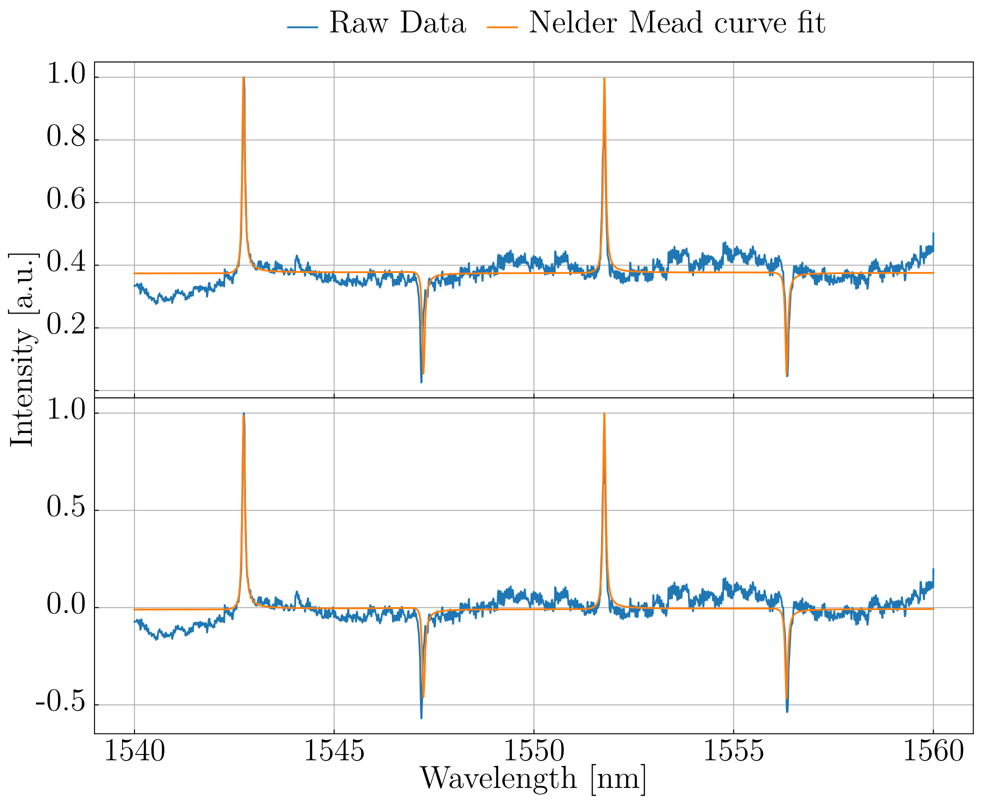


Figure 5.7: Curves fitted using the Nelder-Mead algorithm. In the top image the normalization of data is between 0 and 1, while in the bottom image the data is zero-centered and normalized between -1 and 1.

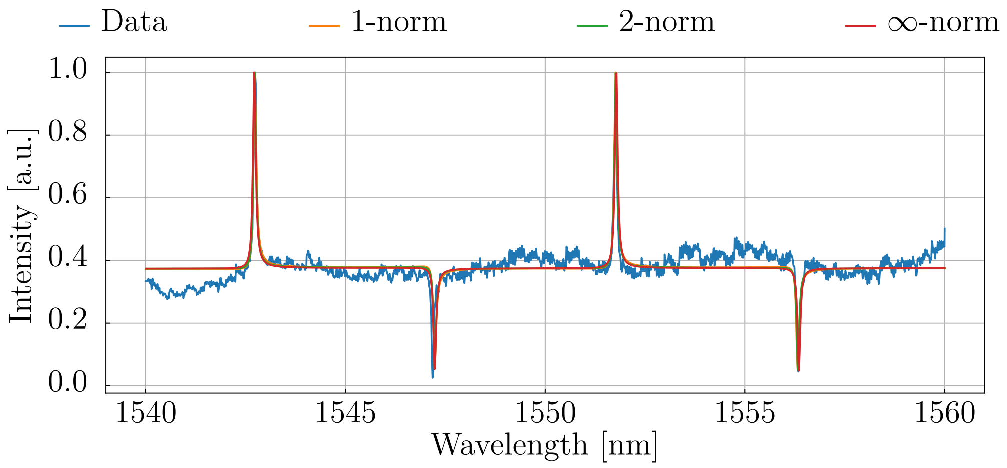


Figure 5.8: Comparison of different norms of the objective function in the Nelder-Mead algorithm. For this measurement, the different norms yield very similar best-fit curves.

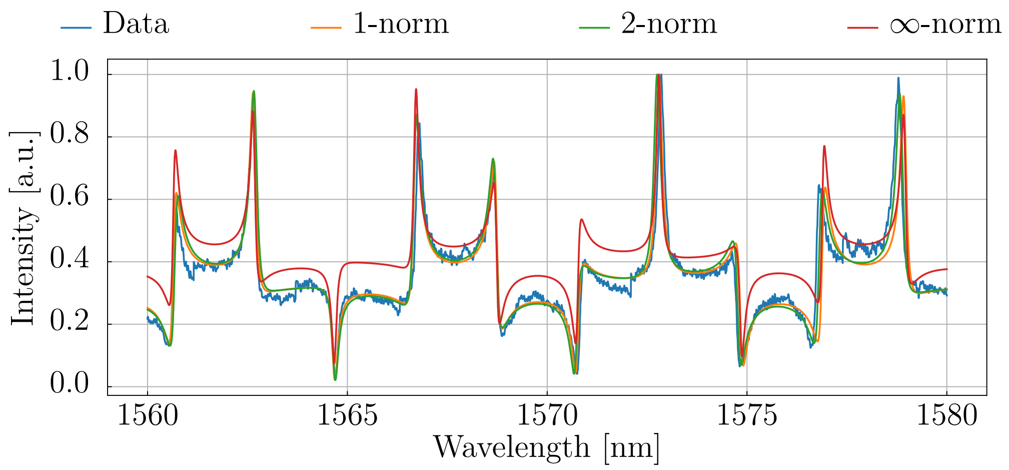


Figure 5.9: Comparison of different norms of the objective function in the Nelder-Mead algorithm. For this measurement, the differences in the fitted curve is more significant depending on the norm used.

cost value if more accurate coefficients are used. It would therefore be of interest to pursue a more detailed investigation to approximate the Sellmeier coefficients more accurately.

Table 5.5: Cost values of curve fits performed using different refractive index aprroximations. The lowest cost value is found when using 6 coefficients.

n_{eff} approximation	No. of coefficients	Cost value
Constant	1	1.620
Sellmeier	2	1.622
Sellmeier	4	1.629
Sellmeier	6	1.616

5.2.6 Residuals

After the preceding considerations, a Nelder-Mead curve fit can be made, using nonzero-centered, the Euclidian norm in the objective function, and Sellmeier approximation of the dispersion. Performing this on sample 1 yields the curve fit, residual plot, and residual distribution shown in figure 5.10. It is immediately apparent that the noise is not normally distributed as expected but follows a sinusoidal pattern. By performing a Fourier transform of the residuals, the sinusoidal pattern is found to have a frequency of $\sim$30 Hz, although the cause for this oscillation in intensity value is unknown. As this value is close to 25 Hz, it is thought this oscillation is a result of electrical noise from the equipment. Poorly isolated wires are close to one another throughout the setup, and can interfere with one another.

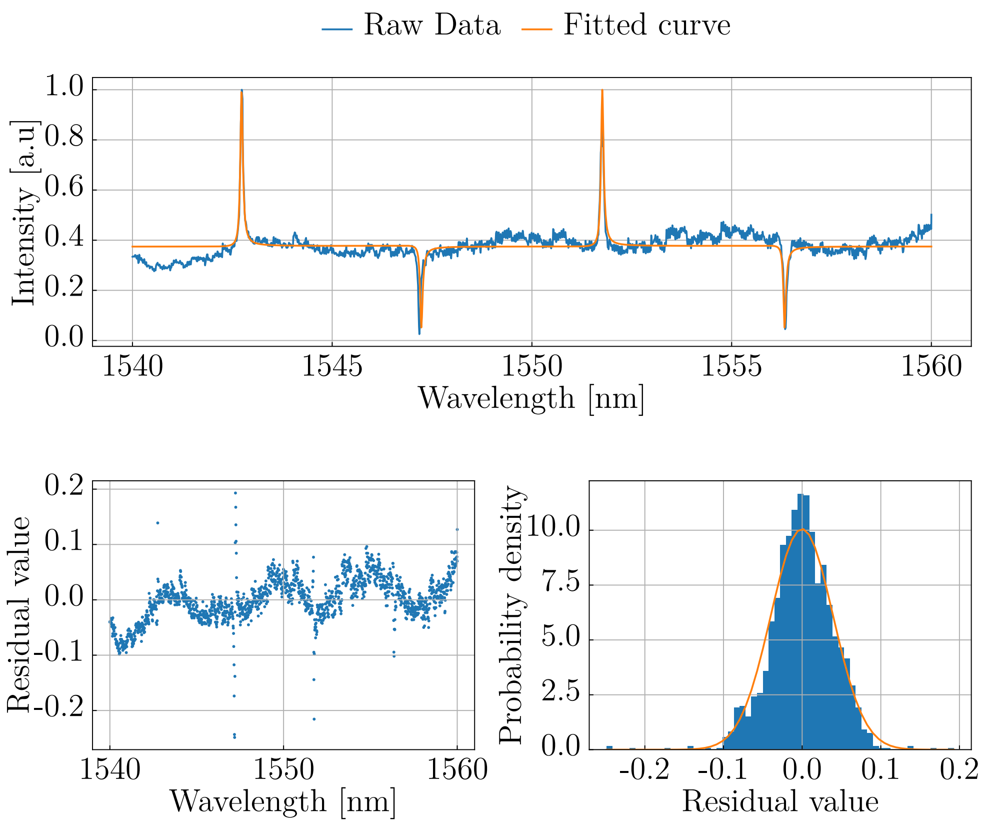


Figure 5.10: Top image shows a good curve fit with low cost-value. The bottom left image shows the residuals plotted against wavelength. The bottom right image shows the probability density against residual value, with a best fit Gaussian curve.

5.3 Resonance Shift in Single Ring MARC

In the experiment examining the resonance shift caused by increasing the saline concentration, Yadav reports a total resonance shift of the signature by 1.45 nm from the original position measured at 0 % to the final measurement made at a saline concentration 18 % [8]. He also calculates a MARC sensitivity to the saline concentration as 80.4 pm/%. By performing the algorithm shown in figure 3.3 on the data sets of sample 2, the resulted resonance shift of the MARC signature is shown in table 5.6. As seen here, the total resonance shift from saline concentrations 0 % to 18 % is calculated to be 1.449 nm, which is off by only 1 pm to the resonance shift calculated by Yadav. Performing a simple linear fit on the resonance shifts and saline concentrations yields a MARC sensitivity of 80.6 %. The reason for the 0.2 % discrepancy here is probably due to the variance caused by the errors included in the calculations by Yadav. The drop port intensities of the MARC at saline concentrations 0 % and 18 % are shown in figure 5.11. Here, it is clearly seen that the whole signature shifts to the right as the saline concentration is increased.

Table 5.6: Parameters for resonance shift for one ring, using the data set provided by Yadav. The effective wavelength shift $\Delta\lambda_{135°,\text{eff}}$ denoted the cumulative wavelength shift from the original position given at saline concentration 0 %

Ring (135°)	$\Delta\lambda_{135°}$[nm]	$\Delta\lambda_{135°,\text{eff}}$[nm]
Water	0	0
6% Saline	0.511	0.511
12% Saline	0.486	0.997
18% Saline	0.451	1.449

.

5.4 Resonance Shift in MARC Containing Two Rings

For the measurements performed on a MARC sensor containing two ring resonators, Yadav presents a more detailed description of the wavelength shift observed than for the single ring MARC. The values provided in the article for the wavelength shift at different saline concentrations are given in 5.7.

As discussed in the article, there is a small variation in the resonance shift of the reference channel, i.e. the ring resonator with angular separation 180°. As the experiment was performed without a temperature controller, it is difficult to determine the exact effects of the temperature in the room on the resonance shift. However, experiments have been done showing wavelength shifts of MARC signatures **??**, where the MARC sensitivity of the temperature is found to be 0.119 nm/°C, assuming a linear relationship between the two variables. A resonance shift of 20 pm then corresponds to a temperature change of 0.17 °C. As this temperature change is

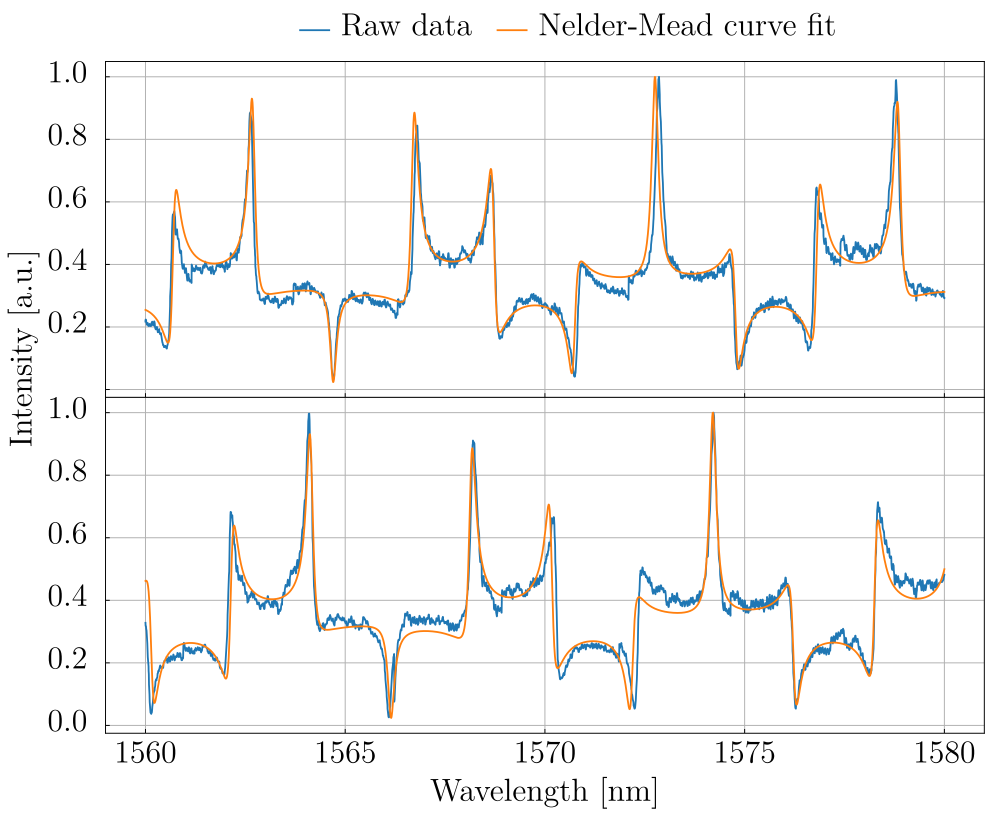


Figure 5.11: Resonance shift of MARC signature caused by an increasing saline concentration. The MARC specifications are the same as those of sample 2, shown in table 5.1. The top plot shows the original signature at 0 % saline concentration, while the bottom plot shows the shifted signature at 18 % saline concentration.

very small, it is no surprise that the reference channel varies between measurements. Therefore, a TEC temperature controller should be considered to ensure that the temperature is consistent between measurements so that the signature position stays the same.

In the article, this reference channel resonance shift is only given approximately as $\sim$ 20 pm. The algorithm used in this work calculates a total resonance shift of 33.2 pm between the first and last measurement. The exact variations at each measurement taken are shown in table 5.7, where it can be seen that the calculated resonance shift varies in both directions. This might not be a bad approximation as the temperature change needed to cause a shift on the pm scale. It is difficult to determine precisely how much the temperature changes between measurements.

For the 135° ring, the algorithm calculated a total horizontal shift of 1.44 nm across the saline concentration difference of 0 % to 18 %, which only differs by 0.05 nm below the value calculated by Yadav.

Table 5.7: Parameters for resonance shift for two rings, using the data set provided by Yadav

Ring (180°)	Ring (135°)	$\Delta\lambda_{180°}$ [nm]	$\Delta\lambda_{180°,\text{eff}}$[nm]	$\Delta\lambda_{135°}$[nm]	$\Delta\lambda_{135°,\text{eff}}$[nm]
Water	Water	0	0	0	0
Water	6% Saline	0.0157	0.0157	0.511	0.511
Water	12% Saline	-0.0448	-0.291	0.489	1.00
Water	18% Saline	0.0332	0.00415	0.445	1.44

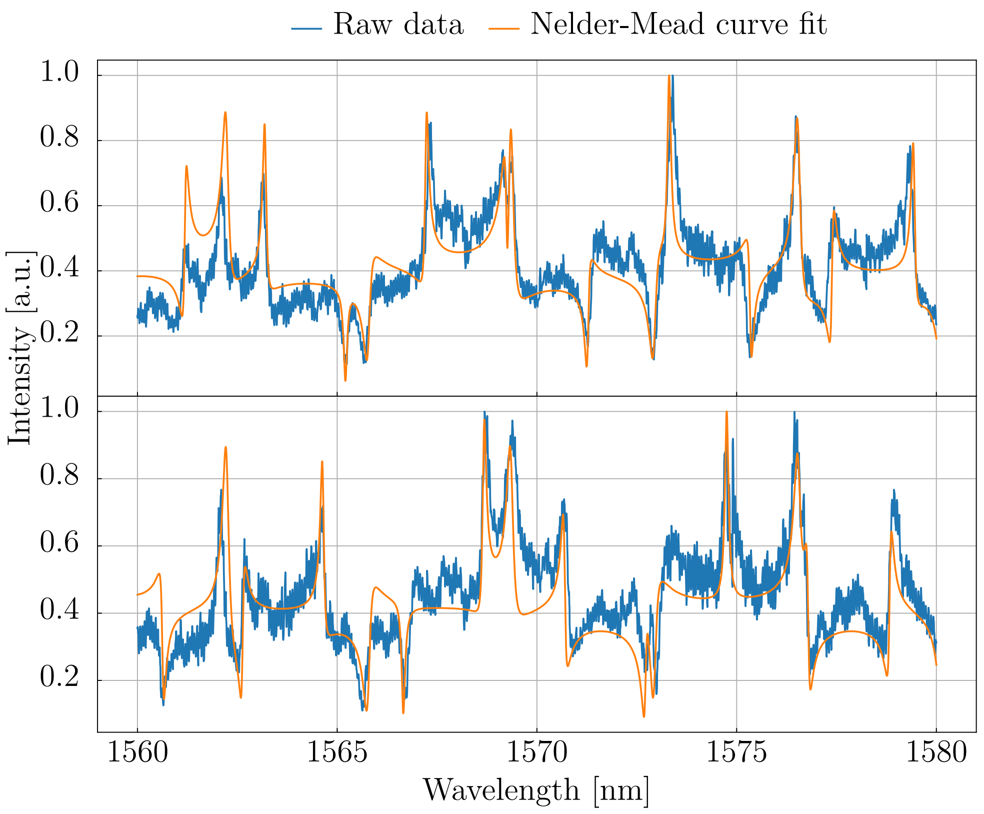


Figure 5.12: Resonance shift caused by increase in saline concentration across a ring with angular separation 135° of the drop port with respect to the throughput port. The sample corresponds to sample 3 in table 5.1. The top image shows the first measurements, where both rings are in contact with only water. The bottom plot shows the final measurement with 18% saline flowing across the 135° ring.

6 | Conclusion

Through the course of this work, two main aspects of transmission measurements of MARC devices have been explored. These are the limitations of the experimental setup used for performing measurements on waveguide structures by the photonics group at NTNU, and the processing of transmission data acquired when performing such measurements. Despite being able to acquire meaningful data using the experimental setup, some factors reducing the reproducibility and accuracy have been identified. These include sources of noise in the data, frictional hysteresis of the actuator in the tunable laser, and the spacing of data points acquired. Interpolation of data points has been considered to mitigate the unequally spaced data points. Additionally, a program has been developed which moves the actuator and performs measurements step-wise, in contrast to the sweep technique previously used.

Curve fitting techniques have been explored to process the data acquired from a transmission measurement as a function of wavelength. This has been achieved by using the various parameters required to calculate the theoretical transmission output of a MARC device as input in two different minimization algorithms. The minimization algorithm will then find optimized parameters which will fit the curve well to the acquired data. Several aspects of such curve fitting have been considered to fit the curves as accurately as possible to the acquired transmission data. These considerations include the normalization of the data, which parameters to use, and the effective refractive index approximation.

Lastly, a program has been developed using the optimized curve fit of the transmission data, which automatically detects the resonance shift in a MARC device. Although the program only has been tested on a resonance shift caused by increasing the saline concentration, the results pair well with the same calculations performed on the same data set of transmission measurements. The values of the calculated resonance shift in this work differ only by 0.05 nm of the same calculations performed by Yadav. The program offers the ability to automatically calculate this shift, as opposed finding this shift manually.

7 | Further Work

For the experimental setup, the next step would be to find ways of better mitigating the noise contributions and frictional hysteresis. A significant improvement to the setup would be to increase the step method speed or synchronize the DAQ and servo motor clocks. This would significantly improve the frequency of the measurement points. Another possibility to improve performance is to utilize a piezoelectric transducer for wavelength sweeping ranther than a servo motor. However, price and availability must be considered, in addition to achieving a similar wavelength range as to the servo motor.

Although the programs developed in this work show promising results, their accuracy and run time can be improved. A thorough experiment to determine the Sellmeier coefficients for the amorphous silicon should be performed to approximate the wavelength-dependent refractive index accurately. Finding these values for the specific material used for the waveguides fabricated by the group would be beneficial. The time spent finding the initial guess of the minimization algorithms is quite long and would benefit from an improvement. If this time is reduced, a natural next step would be to test the method in live sensing.

References

[1] Riley, J. C. "Estimates of Regional and Global Life Expectancy, 1800–2001". *Population and Development Review* 31 (3) (2005) 537–543. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1728-4457.2005.00083.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1728-4457.2005.00083.x.

[2] Prasad, P. N. *Introduction to Biophotonics*. Hoboken, NJ: John Wiley & Sons Inc., 2003. ISBN: 978-0-471-28770-4.

[3] Amano, Y. and Cheng, Q. "Detection of influenza virus: traditional approaches and development of biosensors". *Analytical and Bioanalytical Chemistry* 381 (1) (2005) 156–164. ISSN: 1618-2650. URL: https://doi.org/10.1007/s00216-004-2927-0.

[4] Hussein, H. A. et al. "SARS-CoV-2-Impedimetric Biosensor: Virus-Imprinted Chips for Early and Rapid Diagnosis". *ACS Sensors* 6 (11) (2021). PMID: 34757734 4098–4107. eprint: https://doi.org/10.1021/acssensors.1c01614. URL: https://doi.org/10.1021/acssensors.1c01614.

[5] Ivnitski, D. et al. "Biosensors for detection of pathogenic bacteria". *Biosensors and Bioelectronics* 14 (7) (1999) 599–624. ISSN: 0956-5663. URL: https://www.sciencedirect.com/science/article/pii/S0956566399000391.

[6] Ligler, F. S. et al. "Array biosensor for detection of toxins". *Analytical and Bioanalytical Chemistry* 377 (3) (2003) 469–477. ISSN: 1618-2650. URL: https://doi.org/10.1007/s00216-003-1992-0.

[7] Claes, T. et al. "Label-Free Biosensing With a Slot-Waveguide-Based Ring Resonator in Silicon on Insulator". *IEEE Photonics Journal* 1 (3) (2009) 197–204.

[8] Yadav, M. and Aksnes, A. "Mach-Zehnder interferometer assisted ring resonator configuration for refractive index sensing". *2021 Conference on Lasers and Electro-Optics Europe and European Quantum Electronics Conference*. Optica Publishing Group, 2021 ch_6_5. URL: http://opg.optica.org/abstract.cfm?URI=CLEO_Europe-2021-ch_6_5.

[9] Skrebergene, L. O. "Optimization and fabrication of a fiber-to-waveguide silicon-on-insulator grating coupler". Master's Thesis. NTNU, 2020.

[10] Vos, K. D. et al. “Silicon-on-Insulator microring resonator for sensitive and label-free biosensing”. *Opt. Express* 15 (12) (2007) 7610–7615. URL: `http://opg.optica.org/oe/abstract.cfm?URI=oe-15-12-7610`.

[11] Bahaa E.A. Saleh, M. C. T. *Fundamentals of Photonics*. John Wiley & Sons, Inc., 2019. ISBN: 9781119506874.

[12] Born, M. and Wolf, E. *Principles of Optics*. Cambridge University Press, 2020. ISBN: 978-1108477437.

[13] Rabus, D. G. and Sada, C. *Integrated Ring Resonators, A Compendium*. Springer Nature Switzerland AG, 2020. ISBN: 978-3-030-60130-0.

[14] Yadav, M. et al. “Spectral shaping of ring resonator transmission response”. *Opt. Express* 29 (3) (2021) 3764–3771. URL: `http://opg.optica.org/oe/abstract.cfm?URI=oe-29-3-3764`.

[15] Lu, Y. et al. “Tunable asymmetrical Fano resonance and bistability in a microcavity-resonator-coupled Mach–Zehnder interferometer”. *Opt. Lett.* 30 (22) (2005) 3069–3071. URL: `http://opg.optica.org/ol/abstract.cfm?URI=ol-30-22-3069`.

[16] Gu, L. et al. “A compact structure for realizing Lorentzian, Fano, and electromagnetically induced transparency resonance lineshapes in a microring resonator”. *Nanophotonics* 8 (5) (2019) 841–848. URL: `https://doi.org/10.1515/nanoph-2018-0229`.

[17] Yu, Y. et al. “Fano resonance control in a photonic crystal structure and its application to ultrafast switching”. *Applied Physics Letters* 105 (6) (2014) 061117. eprint: `https://doi.org/10.1063/1.4893451`. URL: `https://doi.org/10.1063/1.4893451`.

[18] Limonov, M. F. et al. “Fano resonances in photonics”. *Nature Photonics* 11 (9) (2017) 543–554. ISSN: 1749-4893. URL: `https://doi.org/10.1038/nphoton.2017.142`.

[19] Avrutsky, I. et al. “Linear systems approach to describing and classifying Fano resonances”. *Phys. Rev. B* 87 (12 2013) 125118. URL: `https://link.aps.org/doi/10.1103/PhysRevB.87.125118`.

[20] Gu, L. et al. “Fano resonance lineshapes in a waveguide-microring structure enabled by an air-hole”. *APL Photonics* 5 (1) (2020) 016108. eprint: `https://doi.org/10.1063/1.5124092`. URL: `https://doi.org/10.1063/1.5124092`.

[21] Xu, H. et al. “Ultra-sensitive chip-based photonic temperature sensor using ring resonator structures”. *Opt. Express* 22 (3) (2014) 3098–3104. URL: `http://opg.optica.org/oe/abstract.cfm?URI=oe-22-3-3098`.

[22] Kim, G.-D. et al. “Silicon photonic temperature sensor employing a ring resonator manufactured using a standard CMOS process”. *Opt. Express* 18 (21) (2010) 22215–22221. URL: `http://opg.optica.org/oe/abstract.cfm?URI=oe-18-21-22215`.

[23] Rajasekar, R., Jayabarathan, J. K., and Robinson, S. “Nano-optical filter based on multicavity coupled photonic crystal ring resonator”. *Physica E:*

Low-dimensional Systems and Nanostructures 114 (2019) 113591. ISSN: 1386-9477. URL: https://www.sciencedirect.com/science/article/pii/S1386947718315601.

[24] Heideman, R. and Lambeck, P. “Remote opto-chemical sensing with extreme sensitivity: design, fabrication and performance of a pigtailed integrated optical phase-modulated Mach–Zehnder interferometer system”. *Sensors and Actuators B: Chemical* 61 (1) (1999) 100–127.

[25] Joaquim R. R. A. Martins, A. N. *Engineering Design Optimization*. Cambridge University Press, 2021. ISBN: 9781108833417.

[26] I. Griva S.G. Nash, A. S. *Linear and Nonlinear Optimization*. Society for Industrial and Applied Mathematics, 2009. ISBN: 978-0-898716-61-0.

[27] F. Gao, L. H. “Implementing the Nelder-Mead simplex algorithm with adaptive parameters”. *Comput Optim Appl* 51 (2010) 259–277. URL: https://link.springer.com/article/10.1007/s10589-010-9329-3.

[28] Kjeldsen, T. H. “History of Convexity and Mathematical Programming”. *Proceedings of the International Congress of Mathematicians (ICM 2010)* (2010) 3233–3257.

[29] J.A. Nelder, R. “A Simplex Method for Function Minimization”. *The Computer Journal* 7 (4) (1965) 308–313.

[30] *Model 300 Series Temperature Controllers Operating Manual*. 325. Newport. 1995. URL: https://www.manualslib.com/products/Newport-325-11240242.html.

[31] *Tunable Laser Kit User Guide*. TLK-L1550M. Thorlabs. 2014. URL: https://www.thorlabs.com/drawings/d12ede35377c89ad-9F0D6E37-B6CE-5C69-5490F17CF64D664D/TLK-L1550M-Manual.pdf.

[32] Paschotta, R. *Article on ’External-cavity Diode Lasers’ in the Encyclopedia of Laser Physics and Technology*. https://www.rp-photonics.com/encyclopedia_cite.html?article=external-cavity%20diode%20lasers. accessed: 17.06.2022.

[33] Littman, M. G. and Metcalf, H. J. “Spectrally narrow pulsed dye laser without beam expander”. *Appl. Opt.* 17 (14) (1978) 2224–2227. URL: http://opg.optica.org/ao/abstract.cfm?URI=ao-17-14-2224.

[34] *Z8 Series Motorized DC Servo Actuators User Guide*. Z812. Thorlabs. 2022. URL: https://www.thorlabs.com/drawings/c8f9bd49e06fa629-D17B3D08-B94E-BA3C-59A361BDD34E917C/Z812-Manual.pdf.

[35] *Manual Fiber Polarization Controllers User Guide*. FPC561. Thorlabs. 2021. URL: https://www.thorlabs.com/drawings/520b3bc2d8397e18-A6FCEB9C-F8F0-0286-1433161B3CCCC935/FPC562-Manual.pdf.

[36] Foundation, P. S. *Python 3.10.0*. 2021. URL: https://www.python.org/downloads/release/python-3100/.

[37] Community, S. *Scipy Nelder-Mead algorithm documentation.* 2022. URL: https://docs.scipy.org/doc/scipy/reference/optimize.minimize-neldermead.html#optimize-minimize-neldermead.

[38] Community, S. *Scipy Nelder-Mead algorithm implementation.* 2022. URL: https://github.com/scipy/scipy/blob/main/scipy/optimize/_optimize.py.

[39] Community, S. *Scipy Least Squares algorithm documentation.* 2022. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html.

[40] Branch, M. A., Coleman, T. F., and Li, Y. "A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems". *SIAM Journal on Scientific Computing* 21 (1) (1999) 1–23. eprint: https://doi.org/10.1137/S1064827595289108. URL: https://doi.org/10.1137/S1064827595289108.

[41] Community, S. *Scipy Least Squares algorithm implementation.* 2022. URL: https://github.com/scipy/scipy/blob/main/scipy/optimize/_lsq/least_squares.py.

[42] Instruments, N. *API package for interacting with the NI-DAQmx driver.* https://github.com/ni/nidaqmx-python/. 2022.

[43] al., T. G. et. *Thorlabs APT.* https://github.com/qpit/thorlabs_apt. 2019.

[44] Nouri, B. "Friction identification in mechatronic systems". *ISA transactions* 43 (2004) 205–16.

[45] Yoon, J. Y. and Trumper, D. L. "Friction modeling, identification, and compensation based on friction hysteresis and Dahl resonance". *Mechatronics* 24 (6) (2014). Control of High-Precision Motion Systems 734–741. ISSN: 0957-4158. URL: https://www.sciencedirect.com/science/article/pii/S0957415814000348.

[46] Numpy. *Numpy Interp.* https://numpy.org/doc/stable/reference/generated/numpy.interp.html. 2022.

[47] Gebrekidan, M. T. et al. "A shifted-excitation Raman difference spectroscopy (SERDS) evaluation strategy for the efficient isolation of Raman spectra from extreme fluorescence interference". *Journal of Raman Spectroscopy* 47 (2) (2016) 198–209. eprint: https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/pdf/10.1002/jrs.4775. URL: https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/abs/10.1002/jrs.4775.

[48] Basnet, K. "Centering of data in Principal Component Analysis in Ecological Ordination". *Tribhuvan University Journal* 14 (1993) 29–34.

[49] Belsley, D. A. "Demeaning Conditioning Diagnostics through Centering". *The American Statistician* 38 (2) (1984) 73–77. eprint: https://www.tandfonline.com/doi/pdf/10.1080/00031305.1984.10483169. URL: https://www.tandfonline.com/doi/abs/10.1080/00031305.1984.10483169.

[50] *Photodiode Amplifier PDA200C Operation Manual*. PDA200c. Thorlabs. 2020. URL: https://www.thorlabs.com/drawings/6c9019cd03e63f82-4AC0AC59-C7A0-F06D-4734CD0166905FC2/PDA200C-Manual.pdf.

[51] *DAQ 6023E/6024E/6025E User Manual*. 6024E. National Instruments. 2000. URL: https://neurophysics.ucsd.edu/Manuals/National%20Instruments/6024E_6024E_6025E%20DAQ%20User%20Manual.pdf.

[52] Liu, H. et al. "A motor-piezo actuator for nano-scale positioning based on dual servo loop and nonlinearity compensation". *Journal of Micromechanics and Microengineering* 13 (2) (2003) 295–299. URL: https://doi.org/10.1088/0960-1317/13/2/318.

[53] Zsurzsan, G. T. et al. "Control and sensor techniques for PAD servo motor drive". *2015 50th International Universities Power Engineering Conference (UPEC)*. 2015 1–5.

[54] Fantoni, A., Lourenço, P., and Vieira, M. "A model for the refractive index of amorphous silicon for FDTD simulation of photonics waveguides". *2017 International Conference on Numerical Simulation of Optoelectronic Devices (NUSOD)*. 2017 167–168.

[55] Salzberg, C. D. and Villa, J. J. "Infrared Refractive Indexes of Silicon Germanium and Modified Selenium Glass∗". *J. Opt. Soc. Am.* 47 (3) (1957) 244–246. URL: http://opg.optica.org/abstract.cfm?URI=josa-47-3-244.

[56] Tatian, B. "Fitting refractive-index data with the Sellmeier dispersion formula". *Appl. Opt.* 23 (24) (1984) 4477–4485. URL: http://opg.optica.org/ao/abstract.cfm?URI=ao-23-24-4477.

A | MARC Intensity Simulation

```python
'''

 This package is converted from a Matlab script provided by Mukesh Yadav.
 Translated and expanded upon by Espen Hovland, 2021, as part of his specialization project at NTNU.
 Some minor changes are added by Nikolai Stensø, 2022, as part of his master's thesis.
 These changes are:
   - Added array compatibility for n_eff
   - Removed MARC dependence on wavelength
   - Added the option to retrieve zero-centered wavelength

'''
import numpy as np
from fractions import Fraction

class Ring:
    """
    Add-drop ring resonator class. Calculates all relevant data when an instance is created.

    Parameters:
        wavelengths          (np.array): Wavelength sweep
        radius               (float):    Ring radius
        angular_separation   (float):    Angular separation of drop- and through-port waveguides
        coupling_coefficient (float):    Self-coupling coefficient of the input waveguide
        loss_coefficient     (float):    Round-trip loss coefficient of the ring
        n_eff                (float):    Effective refractive index of the ring waveguide

    Static methods:
        FSR(ring_radius, n_eff, lambda_0) -> FSR in [m]

    Available data (member variables):
        .r                   (float):    Radius of ring
        .ang_sep             (float):    Angular sep. of through- and drop-port
        .a                   (float):    Round-trip loss coefficient of the ring
        .n_eff               (float):    Effective refractive index
        .t1                  (float):    Self-coupling coefficient of input waveguide
        .t2                  (float):    Self-coupling coefficient of drop-port waveguide
        .fsr                 (float):    Free spectral range
        .fsr_eff             (float):    Effective FSR (single-ring MARC)
        .dp_amplitude        (np.array): Drop-port amplitude response
        .tp_amplitude        (np.array): Through-port amplitude response
        .dp_intensity        (np.array): Drop-port intensity response
        .tp_intensity        (np.array): Through-port intensity response
        .dp_phase            (np.array): Drop-port phase response
        .tp_phase            (np.array): Through-port phase response
    """
    def __init__(self,
                 wavelengths,
                 radius:               float,
                 angular_separation:   float,
                 coupling_coefficient: float    = 0.95,
                 loss_coefficient:     float    = 1,
                 n_eff:                np.array = 3.9) -> None:
        self.r       = radius               # [m]   Radius of ring
        self.ang_sep = angular_separation   # [deg] Angular sep. of drop- and through-port
        self.a       = loss_coefficient     # [1]   Round-trip loss coefficient
        self.n_eff   = n_eff                # [1]   Effective refractive index
        self.t1      = coupling_coefficient # [1]   Self-coupling coeff. of input waveguide
        self.t2      = self.t1 / self.a     # [1]   Self-coupling coeff. of drop waveguide

        phi = 2 * np.pi * self.n_eff * 2 * np.pi * self.r / wavelengths # Round-trip phase shifts
        traversed = self.ang_sep / 360  # Proportion of ring travelled
        act_shift = phi * traversed     # Actual phase shift

        # Amplitude response at drop-port
        self.dp_amplitude = np.asarray(
            (-(np.sqrt(1-self.t1**2) * np.sqrt(1-self.t2**2) * np.power(self.a, traversed)
            * np.exp(1j*act_shift)) / ( 1 - (self.t1 * self.t2 * self.a * np.exp(1j * phi)))),
            dtype=complex
        )

        # Amplitude response at through-port
        self.tp_amplitude = np.asarray(
              (self.t1 - self.t2 * self.a * np.exp(1j * phi))
            / (1 - self.t1 * self.t2 * self.a * np.exp(1j * phi)),
            dtype=complex
        )

        # Intensity response
        self.dp_intensity = np.asarray(np.abs(self.dp_amplitude)**2, dtype=float) # Drop-port
        self.tp_intensity = np.asarray(np.abs(self.tp_amplitude)**2, dtype=float) # Through-port

        # Phase response
        self.dp_phase = np.unwrap(np.angle(self.dp_amplitude)) # Drop-port
        self.tp_phase = np.unwrap(np.angle(self.tp_amplitude)) # Through-port

        # Get the FSR, as if the center frequency is resonant frequency (approximation)
        if isinstance(self.n_eff, np.ndarray):
            self.fsr = self.get_FSR(self.r, self.n_eff[len(wavelengths)//2],␣
↪wavelengths[len(wavelengths)//2])
        else:
            self.fsr = self.get_FSR(self.r, self.n_eff, wavelengths[len(wavelengths)//2])

        # Calculate effective FSR (for single-ring MARCs)
        L    = 1 / traversed
        frac = Fraction.from_float(L).limit_denominator() # Find the reduced fraction N/M
        N    = frac.numerator
        self.fsr_eff: float = N * self.fsr # [nm]

        return

    @staticmethod
    def get_FSR(ring_radius: float, n_eff: float, lambda_0: float = 1550e-9) -> float:
        """
        Calculate the (approximate) FSR.
        Parameters:
            ring_radius (float): [m] Radius of ring
            n_eff       (float): [1] Effective refractive index of ring
            lambda_0    (float): [m] Resonance wavelength of ring.
                                     Defaults to 1550 nm
        Returns:
            Calculated FSR [m]
        """
        return lambda_0**2 / (n_eff*2*np.pi*ring_radius)

class MZI:
    """ Mach-Zehnder interferometer base class """
    def __init__(self, initial_phase: float = 0) -> None:
        self.initial_phase = initial_phase # Phase imbalance of MZI

    @staticmethod
    def get_trans(amplitude, phi_1, phi_2) -> np.array:
        """
        The interference equation, for calculating the amplitude transmittance.

        Parameters:
            amplitude (np.array): The complex amplitude of the signal from the affected arm.
            phi_1     (np.array): The phase of the affected arm.
            phi_2     (np.array): The phase of the unaffected arm.

        Returns:
            t         (np.array): The transmittance of the MZ interferometer.
        """
        t = 0.5*np.exp(0.5j*(phi_1+phi_2+np.pi))*(np.abs(amplitude)*np.exp(0.5j*(phi_1-phi_2))
          + np.exp(-0.5j*(phi_1-phi_2)))
        return t

class MARC(MZI):
    """
    Multi-ring MARC class. Calculates all data upon creating an instance.

    Parameters:
        rings          (Ring):     Rings to include in the MARC
        initial_phase (float):    Phase imbalance from MZI

    Available data (member variables):
        .num_rings    (int):      number of rings in MARC
        .rings        (list):     list of rings in device
        .amplitude    (np.array): output amplitude response of device
        .intensity    (np.array): output intensity response of device
        .zc_intensity (np.array): zero-centered intensity response of device
        .tp_intensity (np.array): through-port intensity response of device
        .phase_output (np.array): output phase response of device
    """
    def __init__(self, *rings: Ring, initial_phase: float = 0) -> None:
        super().__init__(initial_phase=initial_phase)

        self.num_rings  = len(rings)
        self.rings      = [r for r in rings]

        amp_post_rings  = np.zeros(len(rings[0].dp_amplitude), dtype="complex") # Amplitude resp. of all rings
        amp_tp          = np.ones(len(rings[0].dp_amplitude), dtype="complex")  # Amplitude resp. of through-port

        for idx, ring in enumerate(self.rings):
            amp_ring = ring.dp_amplitude # Drop-port amplitude transmission of last ring
            for i in range(0, idx):
                amp_ring *= self.rings[i].tp_amplitude

            amp_post_rings += amp_ring

        for ring in self.rings:
            amp_tp *= ring.tp_amplitude

        amplitude = self.get_trans(amp_post_rings, np.unwrap(np.angle(amp_post_rings)), initial_phase)


        self.amplitude     = amplitude / np.max(np.abs(amplitude)) # Normalize the amplitude
        self.intensity     = np.asarray(np.abs(self.amplitude)**2, dtype=float)
        self.tp_intensity  = np.asarray(np.abs(amp_tp)**2, dtype=float)
        self.phase_output  = np.unwrap(np.angle(self.amplitude))
        zc_intensity       = (self.intensity - self.intensity.mean())/self.intensity.std()
        self.zc_intensity  = zc_intensity/np.max(np.abs(zc_intensity))
        return
```

B | Horizontal shift approximation

```python
import os
from getdata import GetData
from tkinter.filedialog import askdirectory
import copy


directory = askdirectory()
data_list = []
for filename in sorted(os.listdir(directory)):
    data_list.append(GetData(directory +'/'+filename,zc = 0))

data_list[0].no_of_rings, data_list[0].n_eff_method, data_list[0].initial_guess = GetData.ask_initial_guess()
initial_angles = data_list[0].initial_guess[1:3*data_list[0].no_of_rings+1:3]
data_list[0].set_bounds()
if_init = input('Do you wish to initialize the guess? [y/n] ')
if if_init == 'n':
    data_list[0].initial_guess = list(map(float,input("Enter initial guess, each value separated by a space: ").strip().split()))[:len(data_list[0].initial_guess)]
    data_list[0].set_strict_bounds()
else:
    data_list[0].init_nm()
    data_list[0].improve_sellmeier()
data_list[0].find_nm()

x_shift_list = [ [0] for i in range(data_list[0].no_of_rings)]

for i in range(1,len(data_list)):
    data_list[i].interp_to(len(data_list[0].wavelengths))

for i in range(1,len(data_list)):
    data_list[i].no_of_rings   = data_list[0].no_of_rings
    data_list[i].n_eff_method  = data_list[0].n_eff_method
    data_list[i].initial_guess = copy.deepcopy(data_list[i-1].nm_params)

    for ring_no in range(data_list[0].no_of_rings):
        data_list[i].initial_guess[3*ring_no + 2] += x_shift_list[ring_no][i-1]
    data_list[i].set_strict_bounds()
    data_list[i].find_nm()

    for ring_no in range(data_list[0].no_of_rings): # appends the difference in x-shift variable for each ring
        x_shift_list[ring_no].append(
            data_list[i].nm_params[3*ring_no + 2] -
            data_list[i-1].nm_params[3*ring_no + 2]
        )

cumulative_shift = [[] for i in range(data_list[0].no_of_rings)]
for ring_no in range(data_list[0].no_of_rings):
    for i in range(len(x_shift_list[ring_no])):
        if i==0:
            cumulative_shift[ring_no].append(x_shift_list[ring_no][i])
        else:
            cumulative_shift[ring_no].append(x_shift_list[ring_no][i]+cumulative_shift[ring_no][i-1])

for ring_no in range(data_list[0].no_of_rings):
    print("Total resonance shift for ring with angle ",initial_angles[ring_no],": ",sum(x_shift_list[ring_no]))
    print("Cumulative shift array for ring with angle  ",initial_angles[ring_no],": ",cumulative_shift[ring_no])
```

C | Data Analysis

```python
import numpy as np
import pandas as pd
from MARC import MARC, Ring
from scipy import optimize
import numpy as np
from tqdm.contrib.itertools import product
from tkinter.filedialog import askopenfilename
import os
import copy


class GetData:
    """
    Class for inspection of MARC transmission response data.

    Parameters:
        filename           (str): Filename of .txt MARC data file. Datafile should contain two lines. The first line is a list of wavelengths, the second a list of intensities. Separation is <tab>.
        norm               (bool): Boolean value to determine whether the data should be normalized or not. Defaults to 1.
        zc                 (bool): Boolean value to determine whether the data should be zero-centered or not. Defaults to 0.

    Static methods:
        .ask_initial_guess()

    Instance methods:
        .set_file(filename,norm,zc)
        .interp_to(new_length)
        .get_n_eff(wavelengths, *coefficients)
        .marc_curve(parameters)
        .init_guess()
        .residuals(parameters)
        .residuals_ls(parameters,magnification)
        .cost_function(parameters,norm)
        .find_nm()
        .find_ls()
        .find_ls2()

    Private methods:
        ._require(args)
        ._to_ls_bounds()

    Available data (member variables):
        .wavelengths             (np.array): Wavelengths of chosen data file
        .intensities             (np.array): Intensities of chosen data file
        .n_eff_method            (str):
        .no_of_rings             (int):
        .initial_guess           (list):
        .bounds                  (tuple):
        .zero_centered           (bool):
        .normalized              (bool):
        .bounds_confidence       (float):
        .guess_confidence        (float):
        .old_wavelengths         (np.array):
        .old_intensities         (np.array):
        .nm_params               (list):
        .nm_cost                 (float):
        .nelder_mead             (np.array):
```

```python
        .ls_params              (list):
        .ls_cost                (float):
        .least_squares          (np.array):
    """

    def __init__(self,
            filename: str = '',
            norm:    bool = 1,
            zc:      bool = 1,)  -> None:
        self.n_eff_method          = None
        self.no_of_rings           = None
        self.initial_guess         = None
        self.bounds                = None
        self.zero_centered         = zc
        self.normalized            = norm
        self.filename              = filename
        self.bounds_confidence     = 0.5  # percent
        self.guess_confidence      = 0.05 # percent

        self.set_file(filename,self.normalized,self.zero_centered)


    def set_file(self,filename = '', norm: bool = None, zc: bool = None) -> None:
        """
        Reads .txt MARC data file and returns wavelength and intensity as numpy arrays
        Parameters:
            filename            (str): Filename of .txt MARC data file. Datafile should contain two lines. The first line is a list of wavelengths, the second a list of intensities. Separation is <tab>.
            norm               (bool): Boolean value to determine whether the data should be normalized or not. Defaults to 1.
            zc                 (bool): Boolean value to determine whether the data should be zero-centered or not. Defaults to 0.
        Returns:
            wavelengths    (np.array): Numpy array containing the wavelengths.
            intensities    (np.array): Numpy array containing the intensity values.
        """# redo this comment
        if filename == '':
            filename = askopenfilename()
        if norm == None:
            norm = self.normalized
        if zc == None:
            zc = self.zero_centered

        # use pandas to read file. separated by <tab>
        df = pd.read_csv(filename,sep = "        ",header = None)

        #save wavelength and intensity as nunmpy arrays
        wavelengths, intensities = df.iloc[0].to_numpy(),df.iloc[1].to_numpy()

        while wavelengths[-1] == 0 or wavelengths[0] == 0: # adressing a problem where the endpoint wavelengths were 0 in the data files
            if wavelengths[0] == 0:
                wavelengths = wavelengths[1:]
                intensities = intensities[1:]
            if wavelengths[-1] == 0:
                wavelengths = wavelengths[:-1]
                intensities = intensities[:-1]

        # save number of data points at each wavelength into the Series counts
        counts = df.iloc[0].value_counts()

        # checks how many times the first and last wavelength are repeated, then removes all except one
        for i in range(len(counts)):
            if counts.index[i] ==  wavelengths[0]:
                no_starting_wl = counts.iloc[i]
            if counts.index[i] ==  wavelengths[-1]:
                no_final_wl = counts.iloc[i]

        # check if there are duplicate starting or end data points, in the case there are none: keep the original. this could be made better to account for repeat data points in either end, and not just both
        if not no_starting_wl == 1 and not no_final_wl == 1:
            wavelengths = wavelengths[no_starting_wl-1:-no_final_wl+1]
            intensities = intensities[no_starting_wl-1:-no_final_wl+1]



        # check scanning direction. if from high WL to low, removes any WLs over the initial (highest) WL. then flips array so that lowest WL is first
        if wavelengths[0] > wavelengths[-1]:
            entries_to_remove = np.where(wavelengths > wavelengths[0])
            intensities = np.delete(intensities, entries_to_remove , None)
            wavelengths = np.delete(wavelengths, entries_to_remove , None)
            wavelengths = np.flip(wavelengths)
            intensities = np.flip(intensities)

        # if scanning direction is low WL to high, removes any WL below the initial WL
        else:
            entries_to_remove = np.where(wavelengths < wavelengths[0])
            intensities = np.delete(intensities, entries_to_remove , None)
            wavelengths = np.delete(wavelengths, entries_to_remove , None)

        #check if zero-centering is wanted. returns intensities with a mean (close to) zero and a standard deviation (close to) 1.
        if zc:
            intensities = (intensities - intensities.mean()) / intensities.std()

        # check if intensity normalization is wanted. returns the normalized intensities with maximum 1. will keep a mean of 0, but will not keep std of 1.
        if norm:
            intensities = intensities/np.max(np.abs(intensities))
        self.wavelengths = wavelengths
        self.intensities = intensities

    def interp_to(self, new_length):
        # interpolation to new set of data points
        self.old_wavelengths = copy.deepcopy(self.wavelengths)
        self.old_intensities = copy.deepcopy(self.intensities)
        self.wavelengths = np.linspace(self.old_wavelengths[0], self.old_wavelengths[-1], new_length)
        self.intensities = np.interp(self.wavelengths, self.old_wavelengths, self.old_intensities)

    @staticmethod
    def _require(*args):
        for arg in args:
            if arg is None :
                raise ValueError("Not all required parameters for this operation are set. Examples of parameters are .initial_guess and .no_of_rings")

    @staticmethod
    def ask_initial_guess():
        no_of_rings = int(input("How many rings? "))
        initial_guess = []
        # Ask for radius and angle for each ring in device
        for i in range(no_of_rings):
            initial_guess.append(1e-6*float(input("Ring radius (in micrometers) of ring number "+str(i+1)+"? "))) # Ring number i radius guess [m]
            initial_guess.append(     float(input("Ring angle (in degrees) of ring number "+str(i+1)+"? "))      # Ring number i angle guess [deg]
            initial_guess.append(0)                                                                                          # x-shift guess [nm]

        initial_guess.append(0.95)                    # Coupling coefficient [1]
        initial_guess.append(0.98)                    # Loss coefficient guess [1]

        match int(input("Which refractive index approximation? (Enter number) \n [1] Constant\t [2] Cauchy\t [3] Sellmeier\n")):
            case 1:
                n_eff_method = 'constant'
                initial_guess.append(3.9)             # Constant n_eff [1]

            case 2:
                n_eff_method = 'cauchy'
                initial_guess.append(3.89)            # Cauchy A coefficient [1]
                initial_guess.append(0.02402)         # Cauchy B coefficient [1]

            case 3:
                n_eff_method = 'sellmeier'
                match int(input("How many Sellmeier coefficients? (Enter number) \n [2]\t [4]\t [6]\n")):
                    case 2:
                        initial_guess.append(1.34400913e+01) # Sellmeier B_1 coefficient [1] 10.1456
                        initial_guess.append(3.02564647e-03) # Sellmeier C_1 coefficient [1] 0.10611

                    case 4:
                        initial_guess.append(10.1456) # Sellmeier B_1 coefficient [1]
                        initial_guess.append(0.10611) # Sellmeier C_1 coefficient [1]
                        initial_guess.append(0.6378)  # Sellmeier B_2 coefficient [1]
                        initial_guess.append(0.10515) # Sellmeier C_2 coefficient [1]

                    case 6:
                        initial_guess.append(10.6684293)      # Sellmeier B_1 coefficient [1]
```

```python
                        initial_guess.append(0.301516485**2)    # Sellmeier C_1 coefficient [1]
                        initial_guess.append(0.0030434748)   # Sellmeier B_2 coefficient [1]
                        initial_guess.append(1.13475115**2)     # Sellmeier C_2 coefficient [1]
                        initial_guess.append(1.54133408)     # Sellmeier B_3 coefficient [1]
                        initial_guess.append(1104**2) # Sellmeier C_3 coefficient [1]

                    case _:
                        raise ValueError('Invalid refractive index approximation')
            case _:
                raise ValueError('Invalid refractive index approximation')
        return no_of_rings, n_eff_method, initial_guess

    def _to_ls_bounds(self) -> tuple:
        self._require(self.bounds)
        return ([self.bounds[i][0] for i in range(len(self.bounds))],[self.bounds[i][1] for i in range(len(self.bounds))])

    def set_bounds(self):
        self._require(self.initial_guess, self.no_of_rings)

        # Initialize bounds with infinite range
        bounds =  [[-np.inf, np.inf] for i in range(len(self.initial_guess))]

        # Change upper bound of coupling and loss coefficient to 1
        bounds[3*self.no_of_rings][1] = 1
        bounds[3*self.no_of_rings+1][1] = 1

        # Setting bounds on the radii of the rings, to avoid huge changes in minimization algorithms
        for i in range(self.no_of_rings):
            bounds[3*i] = [
                    min(
                    self.initial_guess[3*i] * (1 - self.guess_confidence),
                    self.initial_guess[3*i] * (1 + self.guess_confidence)),
                    max(
                    self.initial_guess[3*i] * (1 - self.guess_confidence),
                    self.initial_guess[3*i] * (1 + self.guess_confidence))
                    ]
        self.bounds = tuple(bounds)                                         # minimization algorithms expect tuples

    def set_strict_bounds(self, strict_val: float = 0.00000000001):
        self._require(self.initial_guess, self.no_of_rings)

        # Setting strict bounds for all variables
        bounds =  [[
                    min(
                    self.initial_guess[i] * (1 - strict_val),
                    self.initial_guess[i] * (1 + strict_val)),
                    max(
                    self.initial_guess[i] * (1 - strict_val),
                    self.initial_guess[i] * (1 + strict_val))]
                    for i in range(len(self.initial_guess))]

        # Allowing infinite range on x-shift
        for ring_no in range(self.no_of_rings):
            bounds[3*ring_no+ 2] =[-np.inf,np.inf]
        self.bounds = tuple(bounds)

    def set_init_bounds(self, strict_val: float = 0.2):
        self._require(self.initial_guess, self.no_of_rings)

        # Setting strict bounds for all variables
        bounds =  [[
                    min(
                    self.initial_guess[i] * (1 - strict_val),
                    self.initial_guess[i] * (1 + strict_val)),
                    max(
                    self.initial_guess[i] * (1 - strict_val),
                    self.initial_guess[i] * (1 + strict_val))]
                    for i in range(len(self.initial_guess))]

        # Change upper bound of coupling and loss coefficient to 1
        bounds[3*self.no_of_rings][1] = 1
        bounds[3*self.no_of_rings+1][1] = 1

        # Allowing larger range on x-shift
        for ring_no in range(self.no_of_rings):
            bounds[3*ring_no+ 2] =[-100,100]
        self.bounds = tuple(bounds)
    def set_init_sellmeier_bounds(self, strict_val: float = 0.1):
        self._require(self.initial_guess, self.no_of_rings)

        # Setting strict bounds for all variables
        bounds =  [[
                    min(
                    self.initial_guess[i] * (1 - strict_val),
                    self.initial_guess[i] * (1 + strict_val)),
                    max(
                    self.initial_guess[i] * (1 - strict_val),
                    self.initial_guess[i] * (1 + strict_val))]
                    for i in range(len(self.initial_guess))]

        # Change upper bound of coupling loss coefficient to 1
        if bounds[3*self.no_of_rings][1] > 1:
            bounds[3*self.no_of_rings][1] = 1
        if bounds[3*self.no_of_rings+1][1] > 1:
            bounds[3*self.no_of_rings+1][1] = 1

        # Allowing larger range on sellmeier coefficients
        for value in range(len(self.initial_guess[3*self.no_of_rings+2:])):
            bounds[3*self.no_of_rings+ 2 + value] =[0,100]
        self.bounds = tuple(bounds)

    def get_n_eff(self,wavelengths, *coefficients):
        # Expecting *coefficients to be of the form *initial_guess[3*no_of_rings+2:]
        self._require(self.n_eff_method)

        # Check which approximation method is chosen
        match self.n_eff_method:
            case 'constant':
                return coefficients*np.ones(len(wavelengths))
            case 'cauchy':
                wavelength = wavelengths*1e6 # m to um
                return coefficients[0] + coefficients[1]/wavelength**2
            case 'sellmeier':
                wavelength = wavelengths*1e6 # m to um
                sum_coeff = 0
                for i in range(0,len(coefficients), 2):
                    sum_coeff += (coefficients[i]*wavelength**2)/(wavelength**2 -␣
↪coefficients[i+1])
                return np.sqrt(1 + sum_coeff)

    def marc_curve(self, parameters):
        self._require(self.no_of_rings,self.n_eff_method)
        ring_list = []
        for ring_no in range(self.no_of_rings):
            shifted_wavelengths            = (self.wavelengths - parameters[ 3*ring_no + 2 ]) *␣
↪1e-9
            ring_list.append(
                    Ring(
                        wavelengths          = shifted_wavelengths,
                        radius               = parameters[ 3 * ring_no ],
                        angular_separation   = parameters[ 3 * ring_no + 1 ],
                        coupling_coefficient = parameters[ 3 * self.no_of_rings ],
                        loss_coefficient     = parameters[ 3 * self.no_of_rings +1],
                        n_eff                = self.get_n_eff(shifted_wavelengths, *parameters[ 3␣
↪* self.no_of_rings + 2:])
                        )
            )

        sensor = MARC( *ring_list )
        if self.zero_centered:
            return sensor.zc_intensity
        else:
            return sensor.intensity


    def init_nm(self, norm = None):
        tmp_min = 100000
        range_list =  []

        # Create ranges to test for x-shift
        for ring_no in range(self.no_of_rings):
            range_list.append(range(-50,50,5))

        # Setting relatively strict bounds on the a priori values
        self.set_init_bounds(0.1)

        # Nested for loop for the different possible x-values
        for x_values in product(*range_list):
```

```python
            for ring_no in range(self.no_of_rings):
                self.initial_guess[3*ring_no + 2] = x_values[ring_no]
            nm_opt = optimize.minimize(
                    fun      = self.cost_function,
                    x0       = self.initial_guess,
                    method   = 'Nelder-Mead',
                    bounds   = self.bounds,
                    args     = (norm)
            )
            tmp_cost          = nm_opt['fun']
            if tmp_cost       < tmp_min:
                tmp_min       = tmp_cost
                tmp_variables = nm_opt['x']

        self.initial_guess    = tmp_variables

    def improve_sellmeier(self,norm = None):
        tmp_min = 100000
        range_list = []

        # Magnify values to get least squares to perform better
        magnification                           = 10**(-np.floor(np.log10(self.initial_guess))+2)

        self.guess_confidence = 0.3
        for value in range(len(self.initial_guess[3*self.no_of_rings + 2:])):
            a = int(magnification[3*self.no_of_rings + 2 + value] * self.initial_guess[3*self.no_of_rings + 2 + value] * (1 - self.guess_confidence))
            b = int(magnification[3*self.no_of_rings + 2 + value] * self.initial_guess[3*self.no_of_rings + 2 + value] * (1 + self.guess_confidence))
            # in an attempt to reduce time spent going through ranges, the invervals in the ranges depend on the length of the range
            range_list.append(range(a,b, int((b-a)/4)))



        self.set_init_sellmeier_bounds(0.0000000001)
        print(range_list)
        print(magnification)
        for coeff in product(*range_list):
            for value in range(len(self.initial_guess[3*self.no_of_rings+2:])):
                self.initial_guess[3*self.no_of_rings + 2 + value] = coeff[value]/magnification[3*self.no_of_rings + 2 + value]
                nm_opt = optimize.minimize(
                        fun      = self.cost_function,
                        x0       = self.initial_guess,
                        method   = 'Nelder-Mead',
                        bounds   = self.bounds,
                        args     = (norm)
                )
                tmp_cost          = nm_opt['fun']
                if tmp_cost       < tmp_min:
                    tmp_min       = tmp_cost
                    tmp_variables = nm_opt['x']
                    print(tmp_variables)

        self.initial_guess    = tmp_variables


    def residuals(self,parameters) -> np.array:
        return self.intensities - self.marc_curve(parameters)

    def residuals_ls(self,parameters,magnification) -> np.array:
        return self.intensities - self.marc_curve(np.divide(parameters,magnification))

    def cost_function(self, parameters, norm = None) -> float:
        return np.linalg.norm(self.residuals(parameters), ord = norm)

    def find_nm(self, norm = None) -> None:
        self._require(self.bounds,self.initial_guess)

        nm_opt = optimize.minimize(
                fun      = self.cost_function,
                x0       = self.initial_guess,
                method   = 'Nelder-Mead',
                bounds   = self.bounds,
                args     = (norm)
        )
        self.nm_params   = nm_opt['x']
        self.nm_cost     = nm_opt['fun']
        self.nelder_mead = self.marc_curve(self.nm_params)
        print("Nelder-Mead fit found.")

    def find_ls(self):
        self._require(self.bounds,self.initial_guess)

        # Magnify values to get least squares to perform better
        magnification                          = 10**(-np.floor(np.log10(self.initial_guess))+1)
        # Keep original x-shift values
        magnification[2:3*self.no_of_rings+3:3] = 1

        least_squares_opt = optimize.least_squares(
                self.residuals_ls,
                np.multiply(self.initial_guess,magnification),
                bounds     = tuple(map(list,np.multiply(self._to_ls_bounds(),magnification))),
                args       = ([magnification])
        )
        self.ls_cost       = np.sqrt(2*least_squares_opt.get('cost'))
        self.ls_params     = np.divide(least_squares_opt.get('x'), magnification)
        self.least_squares = self.marc_curve(self.ls_params)
        print("Least squares fit found.")

if __name__ == '__main__':
    d = GetData(zc=0)
    d.no_of_rings, d.n_eff_method, d.initial_guess = GetData.ask_initial_guess()
    d.set_bounds()
```

D | Motor Control

```python
print("Importing packages takes a little while...")
import nidaqmx
import numpy as np
import thorlabs_apt as apt
import time
import datetime
print("Finished importing packages")

def wavelength_to_pos(wavelength):
    # returns the motor position for a given wavelength. based on the calibration sheet for TLK-
↪L1550M from THORLABS
    #return 1.6+((10.4-1.6)/(1482.24-1618.37)*(wavelength-1618.37))
    #below is the values after calibration by Jens
    return ((6.0663)/(1550-1643.7)*(wavelength-1643.7))

def pos_to_wavelength(pos):
    # returns the wavelength for a given motor position. based on the calibration sheet for TLK-
↪L1550M from THORLABS
    #return 1618.37+((1482.24-1618.37)/(10.4-1.6)*(pos-1.6))
    #below is the values after calibration by Jens
    return 1643.7+((1550-1643.7)/(6.0663)*(pos))

def get_initial_wavelength():
    initial_wavelength = int(input('Enter initial wavelength of sweep [nm]: '))
    if 1450 < initial_wavelength <1650:
        return initial_wavelength
    else:
        print('Wavelength not in range')
        return get_initial_wavelength()

def get_final_wavelength():
    final_wavelength = int(input('Enter final wavelength of sweep [nm]: '))
    if 1450 < final_wavelength <1650:
        return final_wavelength
    else:
        print('Wavelength not in range')
        return get_final_wavelength()

def get_no_of_steps():
    no_of_steps = int(input('Enter number of steps (datapoints): '))
    if no_of_steps > 0:
        return no_of_steps
    else:
        print('Number of steps must be a positive integer')
        return get_no_of_steps()

def get_no_of_measurements():
    no_of_measurements = int(input('Enter number of measurements (files): '))
    if no_of_measurements > 0:
        return no_of_measurements
    else:
        print('Number of measurements must be a positive integer')
        return get_no_of_measurements()

def get_filename():
    filename = input('Enter filename (without file extension): ')
    return filename

def main():
    # set initial parameters for measurement
    #initial_wavelength, final_wavelength, no_of_steps,no_of_measurements,filename = 0,0,0,0,''
    initial_wavelength = get_initial_wavelength()
    final_wavelength = get_final_wavelength()
    no_of_steps = get_no_of_steps()
    no_of_measurements = get_no_of_measurements()
    filename = get_filename()


    # find size of each step to move. might cause extra rounding errors due to moving back and forth between wavelengths and positions so often
    distance_to_move = wavelength_to_pos(final_wavelength) - wavelength_to_pos(initial_wavelength)
    step_size = distance_to_move/(no_of_steps)

    # initiate motor
    motor = apt.Motor(83865265)
    print("Homing started")
    motor.move_home(blocking = True) # use blocking to prevent the rest of the code running until homing is complete
    print("Homing finished")
    print("Current wavelength: ",pos_to_wavelength(motor.position))
    motor.set_velocity_parameters(0,1.5,1.5) # velocity parameter doesn't matter too much when measuring using this script, but probably doesn't need to be max value. to change velocity, change the last number in parantheses. ***** max value = 2.3 *****


    # initiate detector
    task = nidaqmx.Task()
    task.ai_channels.add_ai_voltage_chan("Dev1/ai2")
    try:
        task.start()
    except DaqError: # in case the detector was not properly closed the last time it was used
        task.stop()
        task.start()
    print("Detector successfully connected")
    initial_time = time.time()
    # initiale measurement loop
    for measurement_no in range(no_of_measurements):
        start_time = time.time() # start time for rough estimation of measurement duration

        wavelength_list = []
        amplitude_list = []

        motor.move_to(wavelength_to_pos(initial_wavelength),blocking = True)
        print('Initial position set')
        print("Current wavelength: ",pos_to_wavelength(motor.position))


        # sweeping loop. makes no_of_steps amout of datapoints, moves the motor a tiny distance for each iteration of the loop.
        for i in range(no_of_steps):
            if i == 1: start_time2 = time.time()
            if i == 2: print('Estimated time until sweep is finished: ', datetime.timedelta(seconds  = (time.time() - start_time2)*(no_of_steps-2)*(no_of_measurements-measurement_no))) # only to give a rough estimation of the total duration of the measurement
            moving_pos = wavelength_to_pos(initial_wavelength) + step_size*i # find the next position for the motor
            motor.move_to(moving_pos,blocking = True) # move to next position

            # add wavelength and measured amplitude to separate lists
            wavelength_list.append(pos_to_wavelength(motor.position))
            amplitude_list.append(task.read())


        print("Final position of sweep reached. Current wavelength: ",pos_to_wavelength(motor.position))
        # write wavelength and amplitude lists to file. if several measurements, the filename is appended by "_i" where i is the number of the measurement
        if no_of_measurements == 1:
            f = open(filename+".txt","w")
        else:
            f = open(filename+"_"+str(measurement_no+1)+".txt","w")
        for i in range(no_of_steps):
            if i == no_of_steps -1:
                f.write(str(wavelength_list[i]))
            else:
                f.write(str(wavelength_list[i]) + "\t")
        f.write("\n")
        for i in range(no_of_steps):
            if i == no_of_steps -1:
                f.write(str(amplitude_list[i]))
            else:
                f.write(str(amplitude_list[i]) + "\t")
        f.close()
    print("Total time elapsed: ",datetime.timedelta(seconds = time.time() - initial_time))
    # stop using the intensity measurement task
    task.stop()
    task.close()

    # homing the motor before finishing
    print("Measurements finished, initiating homing")
    motor.move_home(blocking = True)
    print("Homing finished")

if __name__ == "__main__":
    main()
```