Automatic detection efficiency measurements of Superconducting Nanowire Single Photon Detectors

by

Ian Bernabé Maradiaga Rosales

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Thursday July 8, 2021 at 2:30 PM.

Student number:	4876601	
Project duration:	May 1, 2021 - July 8, 2021	
Thesis committee:	Dr. ir. S. F. Pereira,	TU Delft
	Dr. I. Esmaeil Zadeh,	TU Delft
	Dr. A. Adam	TU Delft

Contents

Abstract

Superconducting nanowire single-photon detectors (SNSPDs) are characterized by their quantum limited ability to accurately detect single photons, with low jitter, high detection efficiency and low dark count rate. To achieve this, the detector is cooled to 2-3K, bringing the device in a superconductive state, and is then biased with a direct current (DC) close to its critical current. When a photon impinges the detector, the depairing of Cooper-pairs by the photon leads to local destruction of the superconductivity. The growth of this non-superconducting area, first across and then along the nanowire, leads to the development of a measurable resistance and hence the production of detection pulses. Increasing system detection efficiency (SDE) of detectors has been a long-term goal in the community. Recently ultrahigh efficiency detectors (SDE>98%) have been demonstrated. It has also been shown that the wavelengths dependence of SDE, typically defined by a quarter wavelength cavity, is modulated by fiber-detector airgap (Fabry–Pérot). Measuring such modulations and finding the optimal operation wavelength manually is a time consuming and tedious process. In this thesis a setup for automatic measurement of SDE versus wavelength was developed and benchmarked. For the tested detector, efficiencies were found ranging between 22% and 95% in the wavelengths range between 1260nm and 1650nm. For the optical circuit using Single Mode (SM) fibers only, the automated SDE measurements were unreliable due to shifts in polarisation during measurements. Using PM fibers led to efficiencies very similar to the values measured manually.

1. Introduction

The superconducting nanowire single-photon detector (SNSPD) is on its way of becoming an essential instrument in many applications, including quantum communication [1][2], biomedical imaging [3] and long distance communication [4]. Its capacity to detect single photons with high efficiency [5] [6] [7] [8] [9] short dead time and jitter [10] over a large frequency range are more promising compared to similar technologies, such as avalanche photon diodes (APDs). The latter have reached a plateau of efficiency of around 40% [11]. For the SNSPDs, on the other hand, its community is working towards achieving unity system detection efficiency (SDE) in the infra-red, already achieving 99% SDE recently [12].

Achieving near unit efficiencies consistently will have large impact on various applications, such as quantum communication, biomedical optics and long distance communication.

In quantum cryptography, SNSPDs are mainly applied in quantum key distribution. The detectors are used to encode data into the phase or polarisation of the photons, solidifying communication based on quantum entanglement. Theoretically, it is impossible to break such encryption. [13][14].

In biomedical optics, the study of the lifetimes of biological fluorophores can be of special interest. These are widely used to study cellular and molecular structures, amongst others DNA, proteins and mitochondria. The fluorescent substance has electrons which are able to absorb photons, increasing their energy. Hence, they shortly enter into an excited state before either dispersing their energy or emitting it as a photon, with a lower energy. A lifetime is defined as the time an excited electron takes to emit a photon [15]. The more photons such a substrate will emit, the brighter it will be. Fluorescence analysis yields information about important parameters of the substance, such as the diffusion of a protein, its movements, and its interaction with other molecules. An SNSPD could be used, for instance, to detect small differences in lifetimes and detect a weak number of photons [3].

Using very accurate measurements of photon arrival times, the SNSPD finds an application in long distance communication [4]. Generally, photon counting systems are used to detect weak sources of radiation, i.e. low flux of photons in the visible and near infra-red spectrum. This presented tons of challenges in terms of optimizing the detection rates in high frequency solutions. A high efficiency is required to extract more information encoded in the waves, hence, SNSPD systems can be used.

Many parameters and their interactions influence the efficiency of the SNSPD. They can, however, be roughly divided into two categories: internal efficiency and external efficiency. External efficiency can be split into coupling efficiency and absorption efficiency. Coupling efficiency relates to a possible air gap in the connection between optical fibers and sensors, negatively impacting the efficiency of a detector [12]. Absorption efficiency depends on the absorption coefficient of the material of the detector, and accounts for the fact that not all photons in the cryostat will be absorbed by the nanowire. Internal efficiency relates to how well photons, once inside the system, trigger the detection mechanism correctly. Besides the aforementioned efficiencies, the efficiency of a detector is wavelength dependent, having a

clear peak at a certain wavelength and a clear valley at another wavelength. When designing new SNSPD systems, characterizing its efficiency could be a labour intensive trajectory. Hence, automatisation of this process is critical to accelerate research on SNSPDs.

The goal of this thesis is to automise efficiency measurements on detectors. For this goal, a setup commonly used in characterizing the efficiency of SNSPDs has been employed. The laser source (JGR5 tunable laser), attenuator (JGR Optical Attenuator OA1), power meters (Thorlabs PM100 and Newport 843-R) and drivers were controlled using Python/Matlab for automated measurements to first calibrate the setup and then retrieve efficiency for the connected SNSPD. Python has also been used to automate the analysis, i.e. making graphs, of the retrieved data to characterize the specific detector.

2. Theory

2.1. Detection mechanism

A superconducting nanowire single photon detector (SNSPD) contains a nanowire on a thin film (mostly made of silicon). The detector characterized in this paper is a distributed Bragg detector (DBR). The nanowire (made of $NbTiN$) is placed on two types of alternating materials (SiO_2 and Nb_2O_5, in this case, 6.5 layers deep), on top of a thin film of silicon. At each intersection of material, the light reflects back up to the detector. These types of detectors have a narrow bandwidth of optimal SDE, surrounded by smaller peaks. Another type of frequently used detector is the detector on $1/4$ wavelength $SiO_2/\ Au$. This detector is placed on a thin film (commonly made out of the aforementioned materials) placed on top of a layer with silicon, reflecting the incoming light back up into the detector. These types of detectors have a more broadband range for optimal SDE. A sketch of a DBR detector, a $1/4$ wavelength $SiO_2/\ Au$ detector and their performances are given in figure 1.

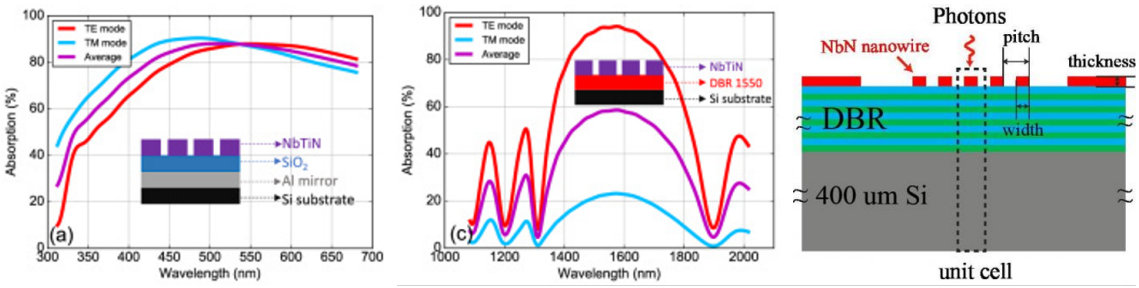

Figure 1: Left: Sketch and simulation of the SDE performance of an $1/4$ wavelength $SiO_2/\ Al$ detector. The shape is broadband, TE is the maximum absorpion and TM is the minimum absorption of photons by the system. Middle: Sketch and simulation of the SDE performance of a DBR detector. The SDE curve has a narrow bandwidth, and evidently the detector is optimimzed for 1550nm. Right: more detailed sketch of a DBR detector. It shows how photons enter the system, the placement of the nanowire (NbN in this case), the structure of the DBR layer (the two colors represent two different materials, in this experiment these were SiO_2 and Nb_2O_5), on top of a film of silicon. Left and middle are adapted from [16]. Right is adapted from [17].

For the detector to function, it should be cooled, often done with liquid helium, to temperatures below its critical temperature, T_c. According to BSC theory, Cooper pairs are formed in this extremely cold state via electron-phonon interactions [18]. These pairs prevent electron-lattice interactions, hence yielding the material superconductive properties. One can only break out of this superconducting state if an external magnetic field is applied (greater than the critical magnetic field) or by heating up the system above the critical temperature T_c. The functionality of the SNSPD is based on the latter property of superconductivity.

A bias current is applied through the nanowire. This current should be slightly lower than the critical current, I_c, which is, the current at which the superconductivity of the system will break. However, since the detector is at a very low temperature and thus has no resistance,

there is no voltage output yet, and the system is still in a superconductive state.

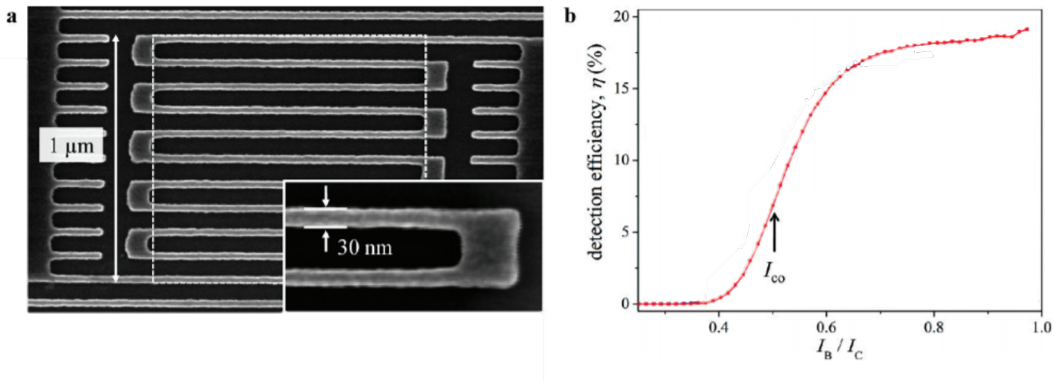

Figure 2: Left: Scanning Electron Microscope image of an SNSPD. Right: the detection efficiency for different bias currents at $\lambda = 1550$ nm in [19], shown to illustrate the influence of bias current on efficiency. Too low and too high of a bias current (not shown) leads to zero detection efficiency. Adapted from [19].

Through an optical fiber, photons can be inputted to the system. If a photon hits the nanowire, it breaks the superconductivity in a small region by depairing a Cooper-pair, releasing heat. The heat builds up, which locally breaks superconductivity in the wire. Then, the heat can be described according to $r * I^2$, known as Joule heating. This means a resistance is developed and the bias current will output a voltage. Afterwards, the current will be divided between the device and the readout circuit, thus reducing the excess heating and letting the system return to its superconductive state. See figure 3. If a bias current is chosen much less than the critical current, an arriving photon can only break the superconductivity in the nanowire locally. However, the area does not grow and therefore the resistive region in the wire will not be there or will not be large enough to give a measurable pulse.

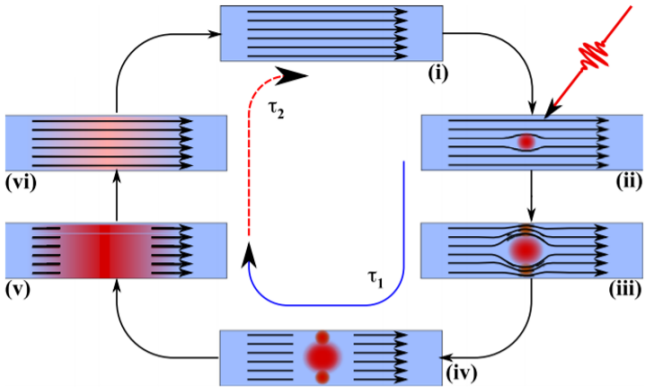

Figure 3: Schematic overview of the detection of photons in SNSPDs. In i), the nanowire is biased with a bias current I_c, slightly below the critical current. At ii), a photon enters the system, yielding a local hotspot region where the superconductivity is broken and releases heat. This heat builds up at iii) and at iv), causing the current to flow around this hotspot. This in its turn creates belts of currents at the edges. Finally, at iv), the heat causes the superconductivity in the nanowire to locally break accross the width of the wire and thus developing a resistance. The current is divided between the device and the readout circuit. This reduces the joule heating and lets heat dissipate to the substrate. The nanowire returns back to its superconductive state as the temperature drops below the critical temperature. From [20].

In figure 4, a simplified version of the readout circuit is shown. The SNSPD is represented in the dashed box with the inductor (superconductors have kinetic inductance) and a parallel connection with a resistor and a switch. If the switch is open, as in the figure below, the schema represents the SNSPD in its resistive state. If the switch is closed, all current flows through the path without the resistor. This represents a superconductive state. Besides the SNSPD, the bias current is shown, as well as the input impedance Z_o and a condensator. The Z_o models the impedance of the transmission line and the condensator is needed for the amplifier (not shown). R_s could be used to influence the dead time of the detector. It was not included in the setup for this experiment.

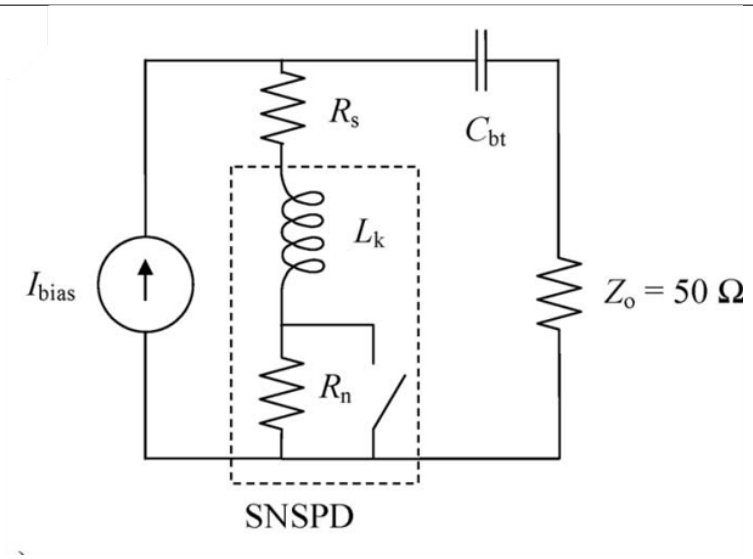

Figure 4: Schematic drawing of the electrical circuit of the SNSPD. The SNSPD is represented as an inductance and a parallel connection with a resistor and a switch. If the switch is open, the detector is in the resistive state. Adapted from [21].

2.2. Performance indicators

Below, a selection of the parameters to indicate performance of an SNSPD are discussed.

2.2.1. Efficiency

In this experiment, efficiency is the parameter of focus. Efficiency is defined as the number of detected photons divided by the total number of photons going into the system. More specifically:

$$\eta_{SDE} = \eta_{coupling}\eta_{absorption}\eta_{internal} \tag{1}$$

$\eta_{coupling}$ is defined as the efficiency of light transmission through coupling different parts of the optical circuit. Every optical part is connected by an optical fiber. At every such intersection, coupling losses will occur. However, the coupling losses at the laser, attenuator and the beam splitter do not affect the efficiency. Only the coupling losses involving the connection between the fiber and the SNSPD matter, since all other losses are corrected for in the calibration measurements. $\eta_{absorption}$ accounts for the fact that not all photons entering the cryostat are absorbed by the nanowire. This term depends on the absorption coefficient of the material, its thickness, polarisation, and the detector optical cavity (thus also the wavelength). These terms can be different when using different materials and thickness and/or different cavity. $\eta_{internal}$ is the ratio between the number of registered detection pulses to the number of photons absorbed by the detector. This term is influenced by choosing the right bias current (DC) and the temperature of the detector and nanofabrication imperfections or constrictions.

2.2.2. Recovery time

After detecting a photon (and thus, after outputting a pulse), an SNSPD cannot detect another photon immediately. The time it takes for the amplitude of the voltage to return to $1/e$ of its peak value is defined as the electrical recovery time. It depends, besides on material properties, on the ratio of the length versus width of the nanowire (a small ratio leads to small recovery times). The exact number differs per detector and can be measured using an oscilloscope, however, typical values are in the tens of nanoseconds. A detection pulse measured with the detector used in this experiment is shown in figure 5.

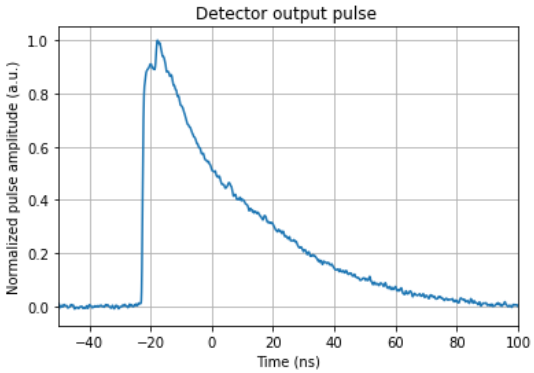

Figure 5: A detection pulse measured with the detector used in this experiment. On the x-axis, the time is displayed in nanoseconds. On the y-axis, the output voltage of the detector has been normalized (thus now in arbitrary units).

2.2.3. Dark counts

Dark counts are the number of registered pulses per second when no light source is directly connected to the detector. This is caused by light not coming from the experimental setup, such as room light or day light, or black body radiation, as well as the intrinsic dark count of the detectors. For this detector, the dark count rate has been measured to be around 250 Hz when the room is dark.

2.2.4. Timing jitter

Timing jitter is the uncertainty in photon arrival times, limiting an accurate determination of the arrival times. It has been shown that under correct conditions, it can reach as low as 3ps [22].

3. Experimental method

To properly measure detection efficiency of an SNSPD detector, multiple setups are required. First, a setup is used to measure power fluctuations of the laser during a period of time. This is done to check for possible drift in laser power, or other types of (structural) fluctuations (see section 3.1). Then, calibration measurements using two power meters are executed, using a setup described in section 3.2. This is done to check what the ratios of the power outputs between the two optical arms are; they should be around 50dB. Under the assumption that these ratios hold true during the SDE measurements, the total number of photons inputted into the system can be calculated if the power is monitored at one of the optical arms. Finally, one of the power meters is replaced by an SNSPD (section 3.3) and the efficiency of the SNSPD system is determined. The measurements in section 3.2 and 3.3 have been executed manually for a set of wavelengths and automised, using only single mode (SM) fibers and using polarisation maintaining (PM) fibers. The benchmark measurements have been executed to get a good indication of the actual performance of the detector, to which the measurements using SM fibers and PM fibers can be compared to (both their manual measurements and their automated ones).

The main goal of this project is to automate detection efficiency measurements on SNSPDs. Hence, a lot of time has been spent on coding and debugging several components of the circuit. In this section, all the aforementioned setups are discussed in a separate part. Besides a description of the actual hardware setup, a global description of the Python scripts used there respectively is also given. All the code referenced to in these sections is given in the appendix, and a directory of all the files and the code is also given on GitHub (see appendix for the full URL). The code was written to be as user friendly as possible.

3.1. Laser Power Fluctuations

To measure the efficiency of an SNSPD detector, the photon input into the system should be known. Moreover, the input should be relatively low (typically around 750.000 photons per second), due to constraints the recovery time (see section 2.2.2) places on the amount of photons the SNSPD can measure. To characterize the SDE of an SNSPD, it is customary to measure its performance for a broad range of wavelengths. For this purpose, the JGR TLS5 Tunable Laser was used. It has a working range between 1260 nanometers and 1650 nanometers, and can be adjusted in increments of 0.1 nanometer for highly accurate measurements, see figure 7.

First, the power of the laser that is incident on the detector is measured for the entire working range of the laser during 20 seconds each, incrementing 10 nm every measurement. This is done to check if there is any drift or other significant deviation in laser power. The laser is directly connected using an optical fiber to the power meter, Thorlabs PM100D, which is in turn connected via an Ethernet cable to the computer. Also, a polarizer is connected in between the laser and the power meter, to compensate for the curling of the SNF28 fiber (SM fiber). These measurements were automated using Python, the code is in the appendix. In figure 6, a schematic drawing of the setup is shown. Figure 7 shows photos of the sketched hardware.

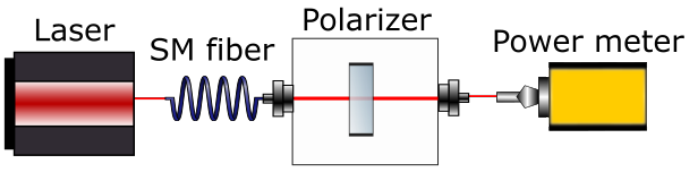

Figure 6: Schematic drawing to measure the laser power using a power meter. The power meter and laser are connected to a computer, to control them remotely (not shown). A polarizer is inserted to compensate for the polarisation shift as a result of the curling of the wire.

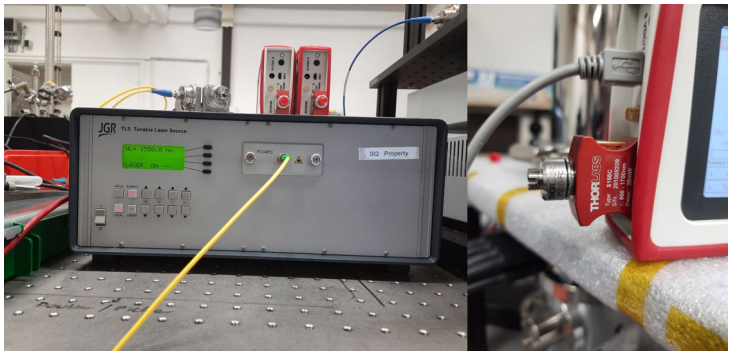

Figure 7: These are the devices used mentioned in figure 6. Left: picture of the laser used (TLS5 tunable laser). Currently, it is set to 1500nm, it has a working range of 1260nm to 1650nm, with a resolution of 0.1nm. Right: Thorlabs PM100 power meter, with an detection error margin of 5% and an accuracy of 1nW.

To retrieve data from the power meter, as in the setup of figure 6, Python libraries could be downloaded online. With the help of these libraries, it is possible to read the power, and set the corresponding wavelength. However, there were some implementation problems, such as not connecting to the power meter or suddenly losing connection. The last issue has not been completely resolved yet. Connecting to and setting the wavelengths of the laser was not an issue. To execute the measurements described in this section, the function $laser\_stability()$ has been written. The user inputs a list of wavelengths he wishes to measure, as well as the number of power measurements per wavelength and a time interval between each measurement. These power measurements are outputted in an Excel file.

The Python function $laser\_stability\_plot()$ has been written to plot the power stability of the laser. It inputs the Excel file described above, and outputs a plot with the power of the laser for all wavelengths. This is done by averaging the powers per wavelength and plotting these values versus its corresponding wavelength. The plot contains error bars, which correspond with the standard deviation in power.

3.2. Calibration measurements

As mentioned above, the power input into the SNSPD system should be around 750.000 photons. If the photon level exceeds this number too much, the recovery time of the system will prevent the SNSPD from measuring a reasonable proportion of the photons, which translates into low (and unreliable) efficiencies. The laser output is, however, many orders of magnitude above this boundary. Therefore, an attenuator is used to achieve this significant reduction. Generally speaking, the power input into the SNSPD system should be 10nW + 50dB attenuation. This extremely low power (in the order of magnitude of femtoWats, 10^{-15}W) cannot be monitored using the Thorlabs PM100 (1nW accuracy) or the Newport 843-R (tens of picoWatts, 10^{-12}). Hence, a beam splitter splits the light into two optical arms with a 50dB attenuation between themselves. During the experiment, the high power arm will then output around 10nW (which can be measured). This experiment used a beam splitter with a ratio of 90%-10%, meaning 90% of the power will enter one arm (arm A) and 10% of the power enters the other arm (arm B) In the low power arm, extra attenuation filters are to be used accounting for the aforementioned extra 50dB attenuation.

To check if the attenuation between these two arms equals 50dB, a set of calibration measurements are executed. This entails connecting the laser to a beam splitter, which has two arms with an attenuation of around 50dB. This value is wavelength dependent, so the real attenuation differs from this value. It has been set to be 50dB at 1550nm for the manual benchmark measurements. After passing the beam splitter, a bench with a polarisation controller is used, which can be set to compensate for the polarization caused by the curling of the optical fibers before entering the beam splitter, as shown in figure 8. This is necessary because not all measurements used polarisation maintaining fibers.

Also, the attenuator has been connected in this setup. This is done to account for the loss in power due to coupling inefficiencies. This is also why the $\eta_{coupling}$ term, discussed in formula 1, only includes the coupling losses involved in connecting the low power arm to the detector. In this experiment, the JGR Optics attenuator is chosen, connected directly to the laser using a SM fiber. It is assumed that the ratio between the power measurements in this calibration measurement is the same ratio once one power meter is replaced by the SNSPD system.

A schematic drawing of the setup for the calibration measurements is shown in figure 8. In figure 9 and in figure 10, pictures are shown of the sketched optical components. First, the optical fibers were all single mode (SM). For another series of SDE measurements, the optical fibers between the Ubenches shown in figure 8 and the fiber to P2, were replaced by polarisation maintaining (PM) fiber.

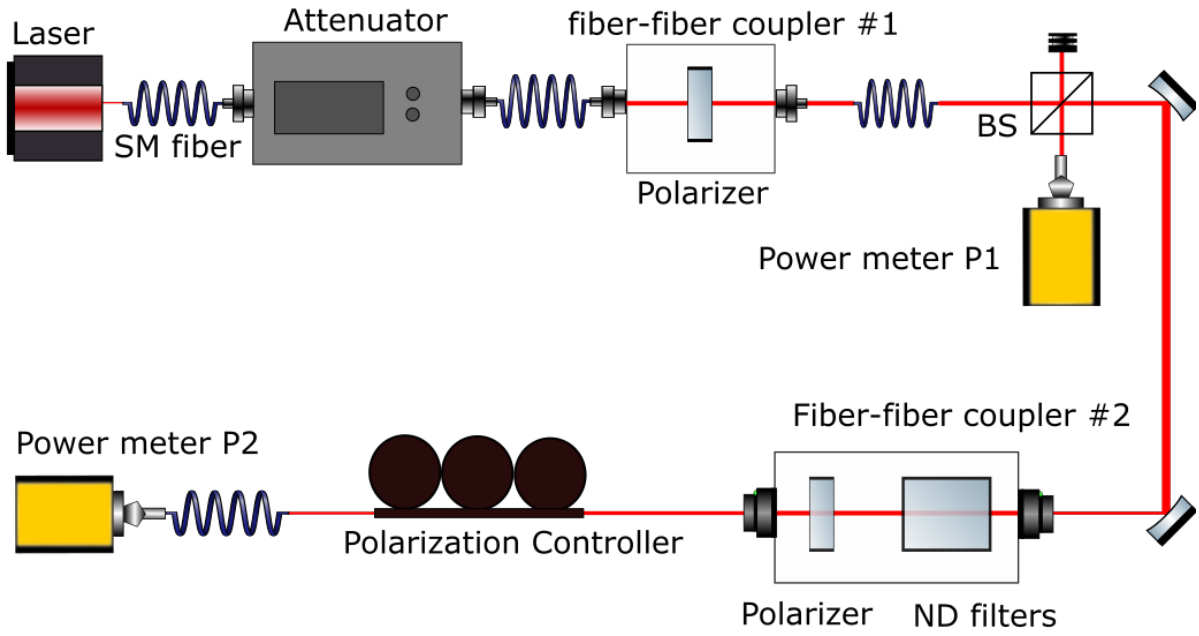

Figure 8: Schematic drawing of the calibration measurements. The laser is connected to the attenuator, then via a Ubench (a fiber to fiber coupler bench) with a polarisation controller, a 90-10 beam splitter with one arm going to a power meter. The other arm first enters a Ubench with attenuation filters to attenuate the ratio between the power outputs of the optical arms to be 50dB at 1550nm. Then, a polarisation controller is connected to compensate for the curling of the wire (no polarisation maintaining fibers were used in one series of measurements) and at last connected to another power meter. For the series of measurements which used polarisation maintaining fiber, the polarisation controller was not needed; instead, a polarizer was rotated along the fast/slow axis to align the polarisation of the light with the polarisation of the detector.

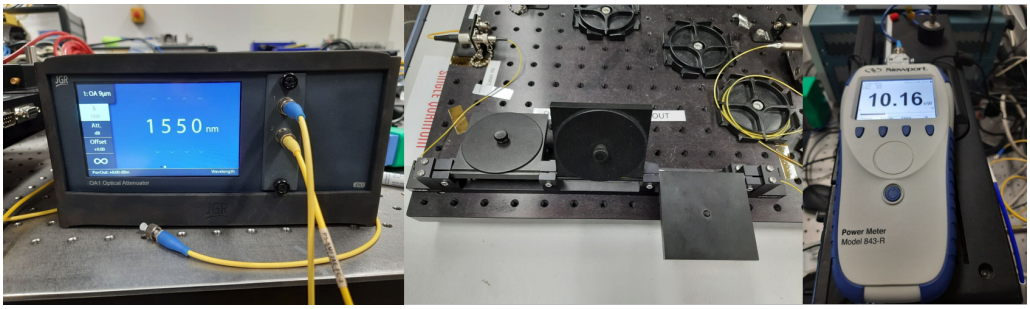

Figure 9: Left: Picture of the attenuator used (JGR optical attenuator OA1). Currently, it is setto attenuate at 1500nm with 0dB, but it can attenuate as much as90dB, with increments of0.01dB. Middle: set of polarisation controllers used to manage the polarisation. For each controller, it is turned in such a way to maximize the number of detected photons. Right: Newport 843-R power meter. It has a 2% accuracy, as compared to a 5% accuracy for the Thorlabs PM100.

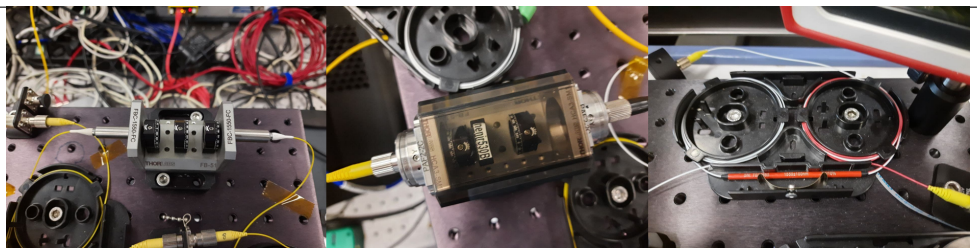

Figure 10: Left: Picture of the fiber-fiber coupler (ubench) $\#1$, only containing polarizers. Middle: Picture of the fiber-fiber coupler $\#2$, containing both polarizer and ND filters (used to attenuate a 50dB ratio between the two optical arms). Right: Beam splitter used. It is a 90-10 beam splitter, officially only for the range of wavelengths $1550 + / - 100$nm.

Evidently, these measurements were automised as well. To communicate with the attenuator, a Python library ($Attenuator.py$) could be installed to set and read the attenuation, as well as the wavelength it is measuring the attenuation at. Just as for the laser, this device also communicates via a GPIB device, and hence the $PrologixGPIBEthernet.py$ package was used.

Then, a function ($calibration()$) was written to perform the calibration measurements. The calibration measurement requires a sweep of the laser through its entire working range. For each wavelength, the attenuator and the power meters are set to operate at this value (although no measurement was performed with the attenuator in this step). 10 measurements are retrieved per wavelength per power meter. The function inputs a list of wavelengths for the setup to be measuring on ($waves$), a number of power measurements per wavelength ($number\_of\_calimeasurements$), and a time interval to wait between measurements. The function outputs Pandas DataFrames containing the power measurements per wavelength for P1 and P2, as well as two Excel files containing the same information for storage.

Once this data is retrieved, it is processed with the Python function $get\_ratios()$. This function calculates the ratios between the measured powers of the calibration measurement per wavelength. It inputs the two Excel files with the calibration data described above, which is sorted per wavelength. First, it calculates the average powers per wavelength, and then it divides these two averages for the ratio. It also converts this ratio to a dB ratio, this number should be around 50dB (as it represents the ratio the beam splitter has been set to). Possibly, this number deviates per wavelength, because the ratio the beam splitter outputs could be wavelength dependent. An Excel file is outputted, containing the average power per wavelength (for both power meters), the ratios and the dB ratios. With these Excel files, the same function $laser\_stability\_plot()$ as used in section 3.1 is used. Now, it is used to plot the fluctuations of the laser power outputted by $P1$ and $P2$, respectively.

3.3. SNSPD efficiency measurements

In this section, an overview will be given of the setup to measure the efficiency of the SNSPD. First, a series of manual measurements has been executed to create a benchmark for the automated measurements. Then, the automated measurements are carried out, and more manual measurements are done. The setup, however is the same, also here the optical fibers were

changed from SM to PM fibers after a series of SDE measurements.

A sketch of the measurement setup to detect the efficiency of an SNSDP detector is shown in figure 11. The JGR TLS5 Tunable Laser is connected through an optical fiber to an attenuator (JGR OA1) to reduce the amount of photons released into the system. First, another JGR optical attenuator has been used, as well as the Thorlabs PM100 power meter. However, it was discovered that the measurements done with the other attenuator were unreliable, since this attenuator reduced the power to such low intensities (probably due to internal power losses) that they could not be measured accurately (the Thorlabs PM100 power meter cannot measure powers below 1nW correctly). Hence, at the last moment, the switch was made to other power meters and were controlled using other Software (Matlab). Due to time constraints, no Python code has been written to control these devices.

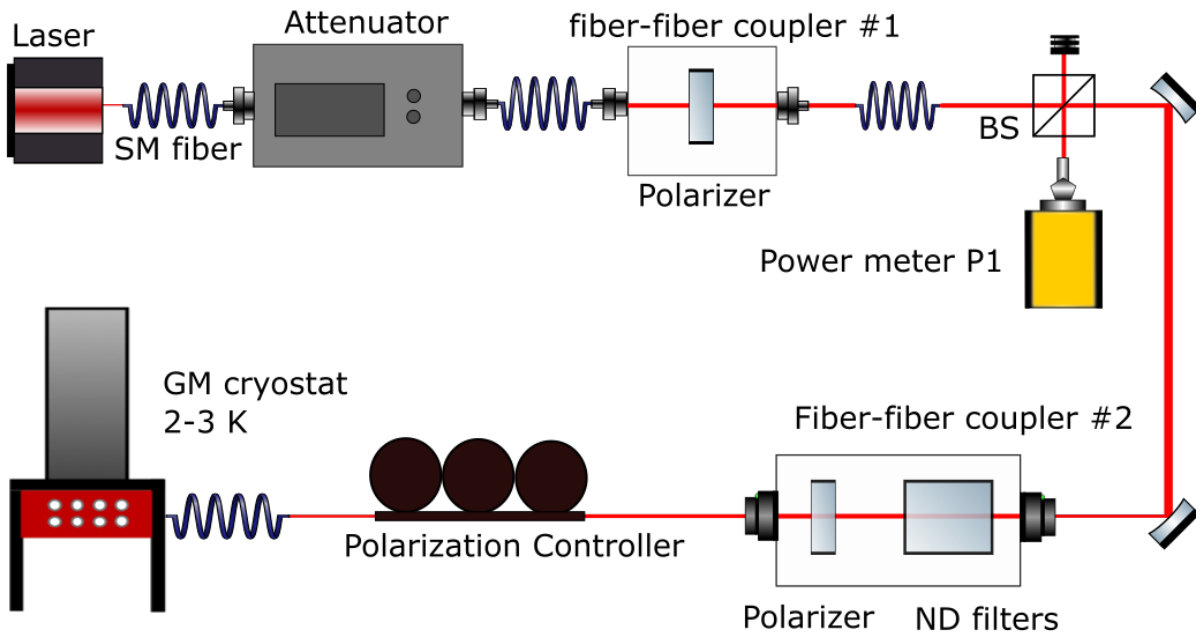

Figure 11: Schematic drawing of the experimental setup to measure efficiency of an SNSDP. The laser is directly connected to the attenuator, and then to a polarisation controller. Then, a beam splitter splits the light in P1 and P2. The lower power arm has a bench to attenuate to a 50dB difference between the arms and a polarisation controller to compensate for the curling of the wire. All elements are connected by optical fibers. The electronic part of the circuit is not included.

After the attenuator, the fiber goes through a beam splitter. The arm with most of the power is connected with a power meter (Newport 843-R, with only 2% error margin) to measure the intensity of the light (and thus, the amount of photons) in the system. Finally, the light goes through a polarization controller before entering the closed cryogenic system (a Gifford-McMahon cryocooler), which has been cooled down to 2.5K using liquid helium and a vacuum. Inside this system, the SNSDP is located, biased with a DC current of around $17\mu A$. Once a photon hits the SNSDP, the superconductivity is broken and an electronic signal is transferred from the detector to the driver, which can then be connected to an oscilloscope or

a computer (as done in this experiment).

The measurement setup is very similar to the setup for the calibration measurements, the most obvious change being the replacement of one power meter by the SNSPD system (see figure 12). When calculating the efficiency, the ratios of the calibration measurements are assumed to be valid during the SNSPD measurements. The power measured in P1 is therefore corrected with this ratio to get the (theoretical) power input (and thus, photon flux) into the SNSPD system. The attenuation of the attenuator has been set in such a way to get a power output on P1 of around 10nW. This corresponds, with the attenuation of the beam splitter of 50dB, with a photon input flux into the SNSPD system of around roughly 780.000 photons at 1550nm.

Figure 12: Picture of the cryostat used in this experiment. The SNSPD is mounted inside. The slots (eight) are also visible. The rest of the setup is the same as in figure 8.

A Python function ($measurements()$) is used to set the laser, attenuator and power meter to a specific wavelength of a list of wavelenghts, and measure the power (10 times per wavelength) and the counts (using the $detected\_counts()$ function described above). This data is stored in Excel files as well. Although this specific code has not been used for this setup, it works with the ThorlabsPM100 power meter and another JGR optics attenuator and has the same working principle as the Matlab used.

For the data processing, Python was still used. The function $photon\_eff()$, is written to calculate the efficiencies. The power sheet measured during the SNSPD counts is inputted, as well as the list with ratios, the list with corresponding wavelengths and the Excel sheet containing the SNSPD counts. This function starts off by calculating the averages of the measured power and uses the corresponding ratio to calculate the power input into the SNSPD system. Then, it calculates the energy of a photon, per wavelength and calculates, using the corresponding power input, how many photons are inputted into the system. This number is used in the denominator of the efficiency, the numerator is taken by the number of detected photons. The efficiencies, system powers, total photon count into the system are outputted in an Excel file for further analysis.

Finally, the function $ploteffwav()$ is made to plot the efficiencies per wavelength. It takes in the efficiency values calculated in the $photon\_eff()$ function and the wavelength.

4. Results and discussion

This section discusses the results in three parts, corresponding with the three parts listed in the experimental method. First, the results of the fluctuations of the laser power are given. Then, the results of the calibration measurements are shown. Finally, the efficiency graphs of the SNSPD measurements are displayed: the manual measurements as a benchmark, the SDE measurements using SM fiber and the SDE measurements using PM fiber.

4.1. Laser Power Fluctuations

To be able to get reliable results of the optical setup, the stability of the laser source is to be checked first. One should make sure there is no drift, or peculiar fluctuations in the laser light itself. The results of these measurements are given in figure 13. Power (in mV) is plotted against the wavelengths (nm), with errorbars. The error bars are the consequence of the uncertainty of the PM100 Thorlabs power meter (5%).

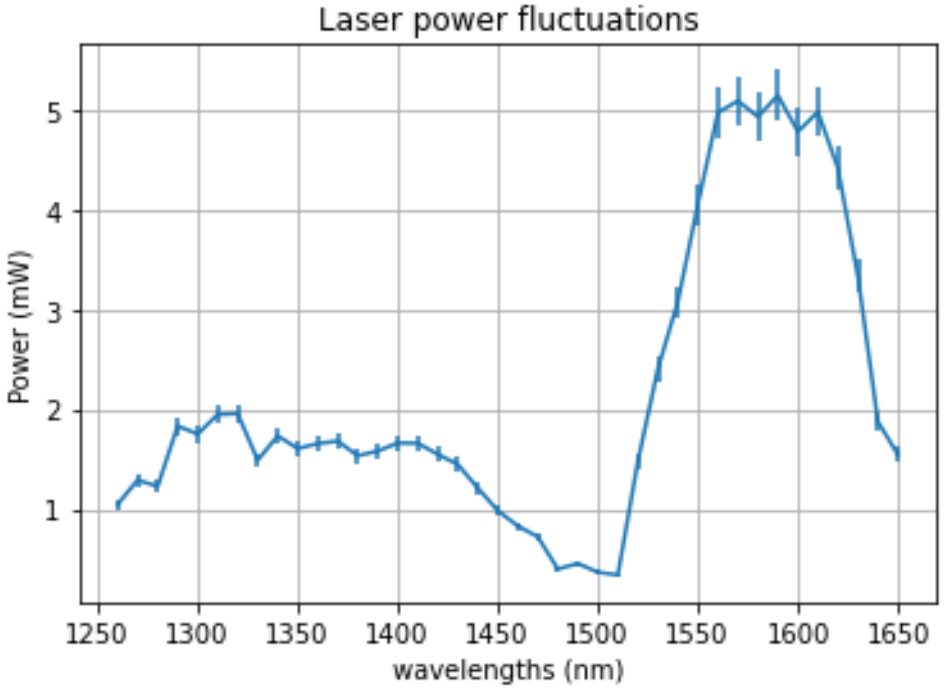

Figure 13: The fluctuations in laser power for the working range of the laser, the error bars are plotted as well. The error comes from the uncertainty of the Thorlabs PM100 power meter (5%).

4.2. Calibration measurements

Here, the results of the calibration measurements are given for all SDE measurements: the benchmark, the measurements with SM fiber and the measurements with PM fiber.

The calibration for the benchmark is done by hand, for a number of wavelengths between 1260nm and 1650nm. Since they were done by hand, the ratio was adjusted in such a way to create an approximate 50dB difference, see figure 14. The laser is connected to the attenuator, and then a beam splitter splits the optical fiber into two arms. Before the light enters the beam splitter and before it enters P2, polarisation controllers are inserted in the optical circuit. P1 is the power meter (Newport) without the attenuation, P2 is the power meter with the attenuation of around 50dB (for 1550nm).

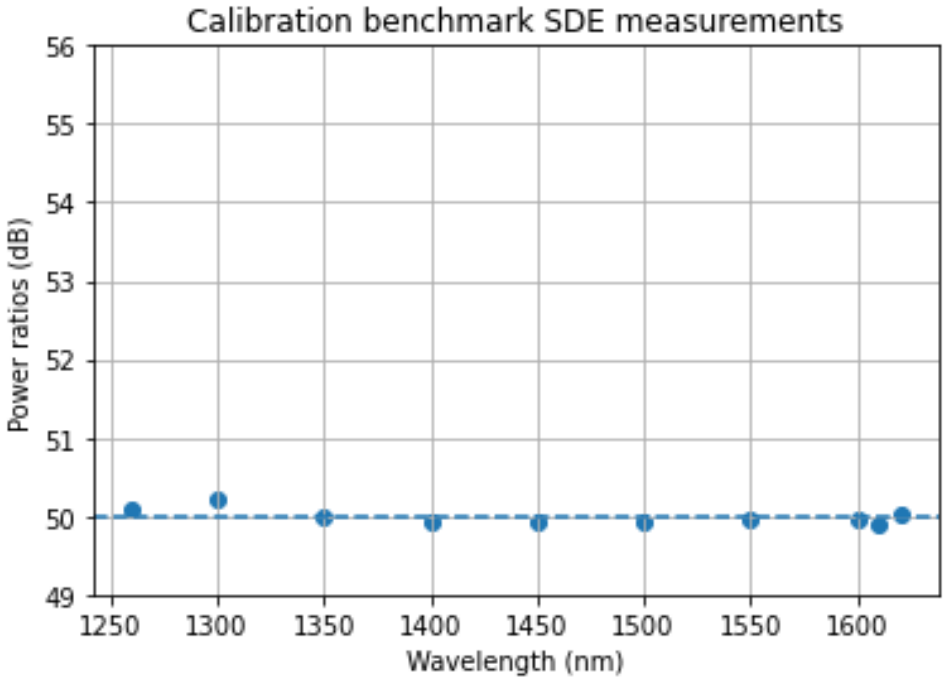

Figure 14: The results from the calibration measurements for the benchmark SDE measurements. Since these measurements were done manually for each wavelength, they are all close to 50dB (dashed line).

In figure 15, plots of the ratios (in dB) between the two power arms are given for the SDE measurements using only SM fiber. These ratios have been measured before starting the SDE measurements, and afterwards. Before measuring the SDE, the calibration has been done in steps of 10nm between 1260nm and 1650nm. Afterwards, the calibration was done in steps of 20nm for the same range of wavelengths as mentioned before. As can be seen, the ratios differ per wavelength, but for 1550nm, it has been set to 50dB (before measuring SDE). However, the calibration ratios measured before determining the SDE differ significantly from the calibration ratios measured after the experiment. This means the measurements done with the SM fibers were unreliable. The deviations in these measurements could be explained by a different polarisation during each of these measurements. This could be cause by touching the fiber between the two ubenches described in figure 8. The input fiber probably has no role in this, since it first goes through a polariser. Besides, there is error as a result of coupling and decoupling of the fiber, although this cannot explain these huge differences. After calibrating the first time, the fiber to P2 is disconnected and connected to the SNSPD. After the experiment, the fiber is reconnected to power meter 2 (P2). Because the air gap between

the (power) sensor and the fiber varies when reconnecting, the connection is never the same. This results in deviations in power.

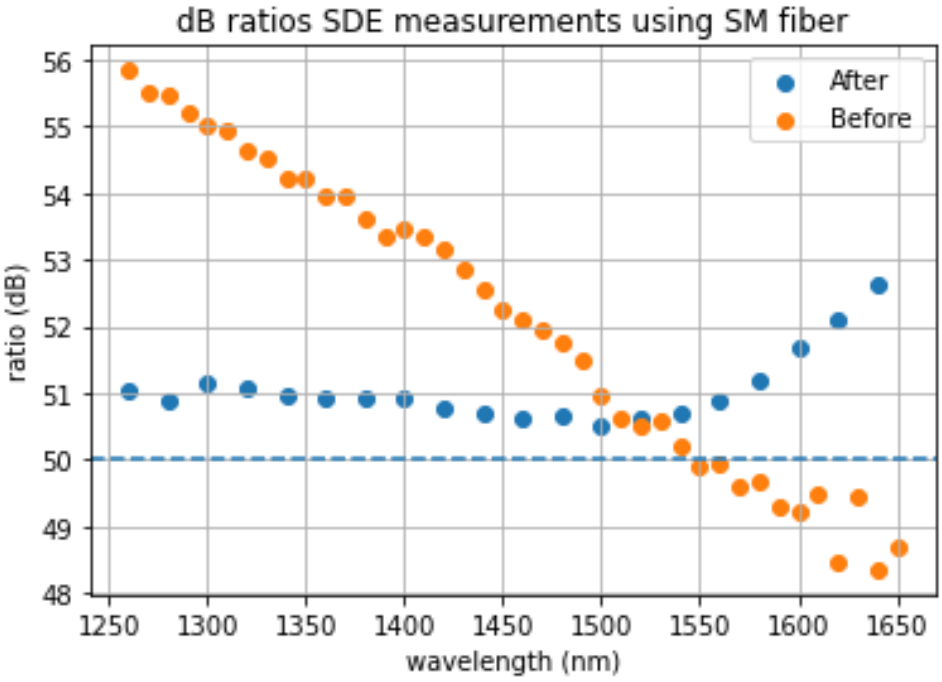

Figure 15: Plot with the ratios (in dB) between P1 and P2 for wavelengths between 1260nm and 1650nm. The calibration has been done twice, before the SDE measurements (in steps of 10nm) and after the SDE measurements (in steps of 20nm). The dashed line is the ideal ratio between the two power meters, which is achieved at 1550nm - as can be seen in the calibration taken before the SDE measurements. As can be seen, the ratios have shifted during the experiment. This could be explained by a shift in polarisation, because a (SM) wire has been touched.

Below, the measurements with the PM fiber are given. Due to time constraints, the ratio has not been set to be exactly 50dB at 1550nm. These calibration measurements have been done after the SDE measurements, in the range of 1300nm to 1650nm, with increments of 50nm (plus a calibration measurement at 1260nm, the starting point of previous measurements). Since the fibers used were polarisation maintaining, a shift of polarisation during SDE measurements will not be of greatest concern in this case. In this case, the error as a result of coupling and decoupling could be a problem. As can be seen in the figure, the point at 1550nm is a significant outlier to the trend. Unfortunately, no data is available of the ratio before the measurement to check if this ratio has shifted, but as the fibers used were polarisation maintaining the chances are lower this has happened and it has been assumed that the ratios stayed constant. The point being outlier has no influence on the efficiency in itself.

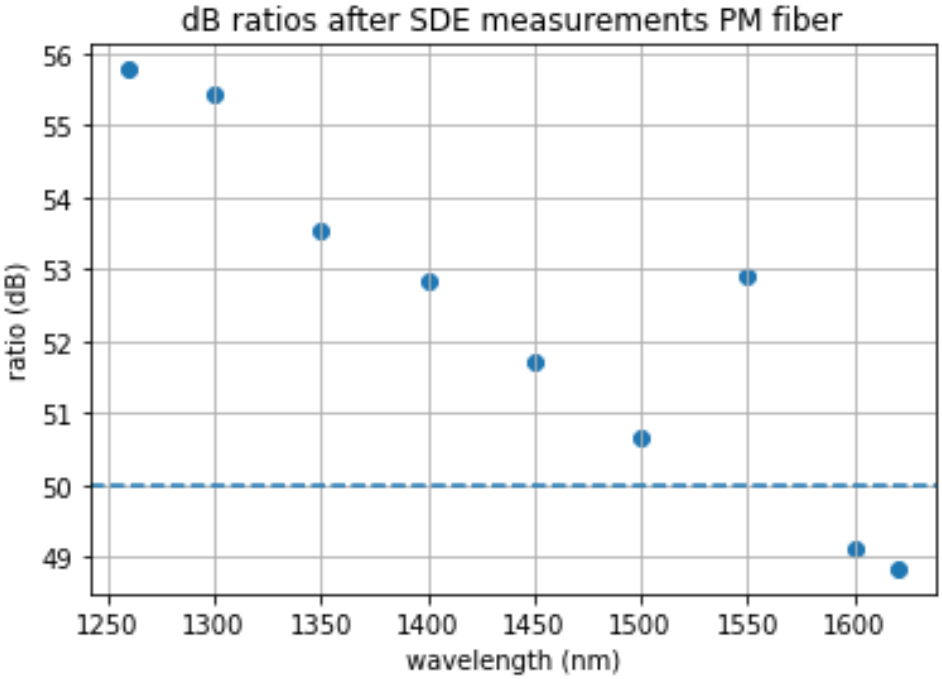

Figure 16: Plot with the ratios (in dB) between P1 and P2 for wavelengths between 1260nm and 1650nm, in steps of 10nm. The dashed line is the ideal ratio between the two power meters, which is achieved at 1550nm. These measurements were taken after the SDE measurements were finished, with PM fiber in the optical circuit.

4.3. Detector efficiency measurements

In this section, the results of SDE measurements are given. First, the results of the manual SDE measurements are given. In the two other sections, the SDE measurements using SM fibers and using PM fibers are displayed, respectively.

The manual SDE measurements can be used as a benchmark for the other aforementioned SDE measurements. As can be seen in figure 17, the efficiency of the detector has been measured for several wavelengths between 1260nm and 1650nm, ranging between 22% and 95%. If compared to the typical behavior of a DBR detector, as shown in figure 1 (middle figure), it looks as expected for the lower wavelengths. For the upper wavelengths (around 1600nm), the SDE is expected to be lower and follow a more downward trend. Although the ratio between the two optical arms was reasonable during calibration measurements, see figure 14, it could be that the ratio has shifted during measurements as a result of a shift in polarisation (after all, SM fibers were used). Another error (for all datapoints) is the change in ratio due to the decoupling of the fiber from P2 to the SNSPD, as every coupling is unique and results in a slightly different power loss.

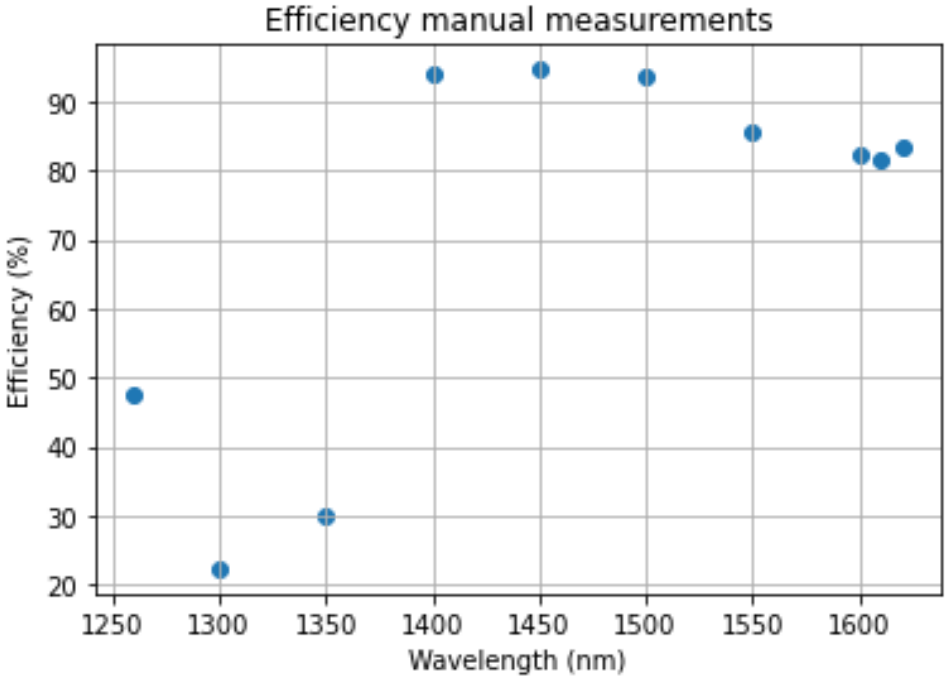

Figure 17: Efficiency plot for the mounted detector in the cryostat. These measurements were taken manually. It has data points between 1260nm and 1650nm, with efficiencies ranging between 22% and 95%.

4.3.1. SDE measurements using SM fibers

In this section the results of the SDE measurements using SM fibers are given, see figure 18. As mentioned above, the calibration using SM fibers was not reliable. A calibration was done before the measurement ("ratios A") and another calibration was done after the measurement ("ratios B"). These huge deviations and unrealistic efficiency values can be explained by the unreliable calibration measurements, as a result of a shift in the polarisation. From this figure, one could say that ratios B are closer to reality than ratios A. This could also be determined from figure 15: the ratios taken after the measurement are closer to the desired value of 50dB. Still, even these values are not reliable, since they may not represent the ratios which were true during the measurements.

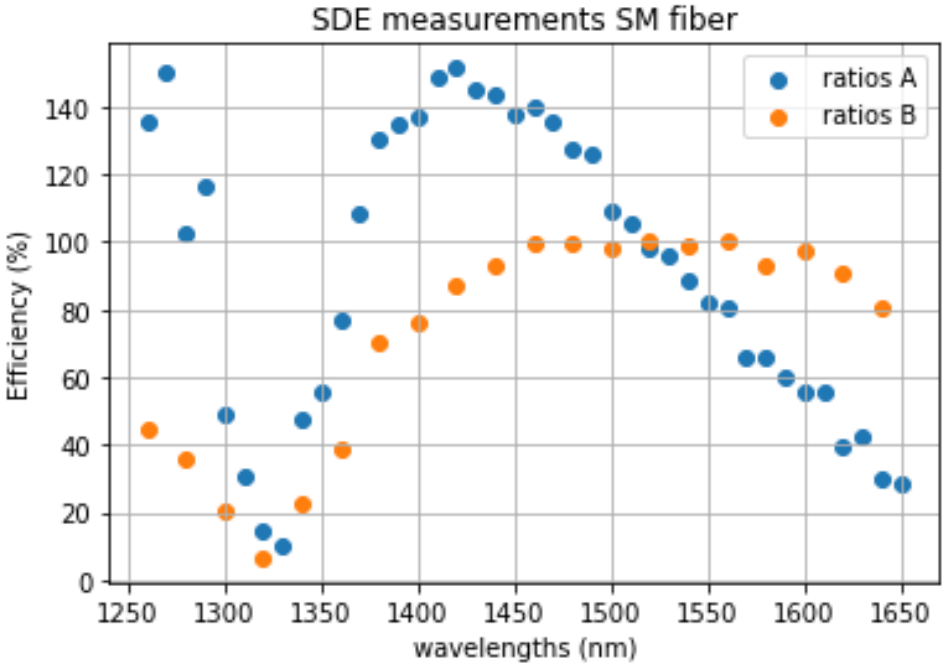

Figure 18: Efficiency plot for the mounted detector in the cryostat. These measurements were taken automatically using software. It has data points between 1260nm and 1650nm, with efficiencies ranging between 8% and 150%. The values shown here are unreliable, as the calibration measurements used for both sets do not represent the correct ratio of power in the optical arms during the SDE measurements. The measurements were taken between 1260nm and 1650nm, with a 10nm stepsize for ratios A and a 20nm stepsize for ratios B.

4.3.2. SDE measurements using PM fibers

In this section, the results of the SDE measurements using PM fibers are given, see figure **??**. A series of manual measurements and automated measurements were taken. As can be seen in the figure, the measurements match the expected shape of a DBR detector, and agree with the benchmark set in figure 17. However, there is a slight mismatch between the series of manual SDE measurements and automated SDE measurements. The manual measurements agree better with the set benchmark than the automated measurements. This could be explained by the coupling and decoupling of the optical fiber mentioned above.

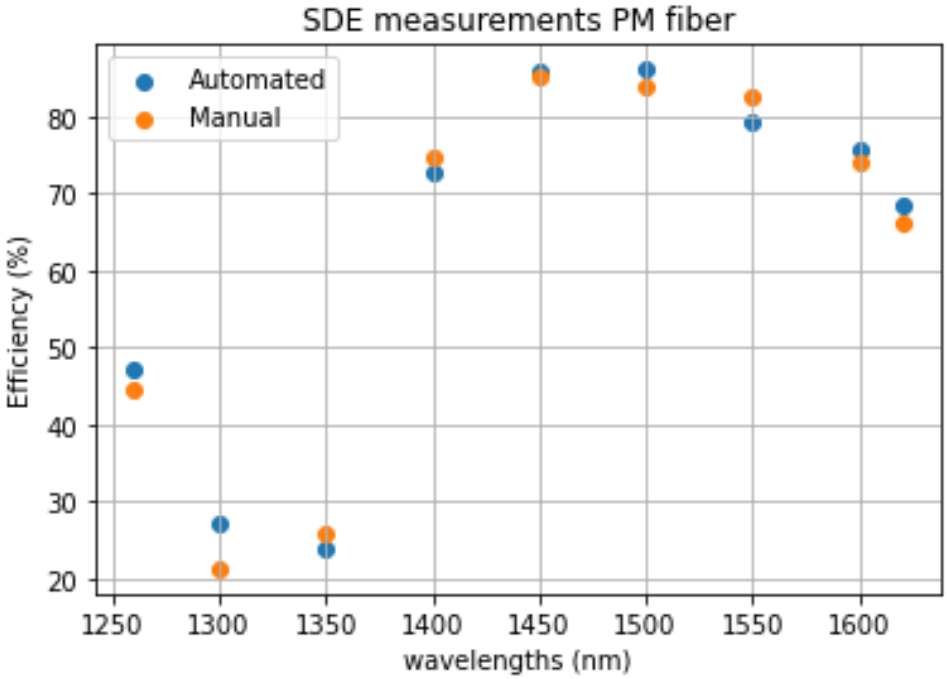

Figure 19: Efficiency plot for the mounted detector in the cryostat. These measurements were taken manually. It has data points between 1260nm and 1650nm, with efficiencies ranging between 22% and 95%.

5. Conclusion and recommendations

The goal of this thesis was to design a precise measurement setup to automatically perform efficiency measurements on SNSPDs. First, a measurement setup was designed to accurately monitor the low photon input into the SNSPD. This has been done by connecting a laser first to an attenuator and then to a beam splitter. From here two optical arms are leading to i) a power meter (unattenuated by the beam splitter) and ii) to another power meter during calibrations or to the cryostat (attenuated by the beam splitter). Polarisation controllers were inserted before the beam splitter and before the SNSPD to compensate for unwanted polarisation, and attenuation filters were used in the low power arm to achieve a 50dB ratio at 1550nm between the powers of both optical arms. Calibration measurements were executed to determine the ratio between the arms accurately for the specified range of wavelengths (1260nm - 1650nm). Then, the laser power was attenuated to output around 10nW on arm i), to achieve a reasonable photon influx to the SNSPD. All measurements were automated and processed using Python. Measurements have been done using SM fibers throughout the optical circuit, and then polarisation maintaining fibers were used.

5.1. Conclusion

When determining the efficiency of the detector, it was found that it depends for a large part on the wavelength in this setup. The measurements with the SM fiber turned out to be unreliable, as the polarisation has shifted during the experiment. The SDE measurements with polarisation maintaining fiber matched the benchmark set much better. The mismatches with the manual benchmark data could partly be explained by the coupling and decoupling of the fiber in switching from calibration measurements to SDE measurements. The Python code, which was written for this project, worked properly during the experiment: communicating with all the hardware in the optical circuit and then processing the data properly into Excel files and graphs.

5.2. Recommendations

For further research, it is recommended to use polarisation maintaining fibers. Moreover, the ratio between the optical arms can be set closer to an attenuation of 50dB, or even replaced by a beam splitter specifically designed for the range of wavelengths of the measurements. Also, the Thorlabs power meter can be replaced by a more sensitive power meter (such as the Newport 843-R) in further research for more accurate restults.

Acknowledgements

I would like to thank Silvania Pereira and my supervisor Iman Esmaeil Zadeh for the opportunity to do my Bachelor Final Project at the Optics and Quantum Optics department, here at the TU Delft. Especially, I would like to thank Iman Esmaeil Zadeh, Jin Chang and Niels Los from Single Quantum for their availability to answer all my questions and help me out in the lab. Finally, I would like to thank Single Quantum for the equipment which has been lent for this experiment.

References

[1] Juan Yin, Yuan Cao, Shu-Bin Liu, Ge-Sheng Pan, Jin-Hong Wang, Tao Yang, Zhong-Ping Zhang, Fu-Min Yang, Yu-Ao Chen, Cheng-Zhi Peng, and Jian-Wei Pan. Experimental quasi-single-photon transmission from satellite to earth. *Opt. Express*, page 21, 2013.

[2] Giuseppe Vallone, Daniele Dequal, Marco Tomasin, Francesco Vedovato, Matteo Schiavon, Vincenza Luceri, Giuseppe Bianco, and Paolo Villoresi. Interference at the single photon level along satellite-ground channels. *Phys. Rev. Lett*, page 116, 2016.

[3] O. Marinov M. J. Deen D. Palubiak, M. M. El-Desouki and Q. Fang. High-speed, single-photon avalanche-photodiode imager for biomedical applications. *IEEE Sensors Journal*, 2011.

[4] Yury Lobanov, Michael Shcherbatenko, Alexander Semenov, Oliver Kahl Vadim Kovalyuk, Simone Ferrari, Alexander Korneev, Roman Ozhegov, Natalia Kaurova, Boris M. Voronov, Wolfram H. P. Pernice, and Gregory N. Gol'tsman. Superconducting nanowire single photon detector for coherent detection of weak signals. 2017.

[5] Iman Esmaeil Zadeha, Johannes W. N. Los, Ronan B. M. Gourgues, Violette Steinmetz, Gabriele Bulgarini, Sergiy M. Dobrovolskiy, Val Zwiller, and Sander N. Dorenbos. Single-photon detectors combining high efficiency, high detection rates, and ultra-high timing resolution. *APL Photonics*, 2017.

[6] WeiJun Zhang, LiXing You, Hao Li, Jia Huang, ChaoLin Lv, Lu Zhang, XiaoYu Liu, Jun-Jie Wu, Zhen Wang, and XiaoMing Xie. Nbn superconducting nanowire single photon detector with efficiency over 90 % at 1550 nm wavelength operational at compact cryocooler temperature. *Science China Physics, Mechanics Astronomy volume*, 2017.

[7] F. Marsili, V. B. Verma, J. A. Stern, S. Harrington, A. E. Lita, T. Gerrits, I. Vayshenker, B. Baek, M. D. Shaw, R. P. Mirin, and S. W. Nam. Detecting single infrared photons with 93% system efficiency. *Nature photonics*, 2013.

[8] Dileep V. Reddy, Adriana E., SaeWoo Lita, Richard Nam, P. Mirin, and Varun B. Verma. Achieving 98% system efficiency at 1550 nm in superconducting nanowire single photon detectors. *Rochester Conference on Coherence and Quantum Optics*, 2019.

[9] Iman Esmaeil Zadeh, Johannes W. N. Los, Ronan B. M. Gourgues, Jin Chang, Ali W. Elshaari, Julien Romain Zichi, Yuri J. van Staaden, Jeroen P. E. Swens, Nima Kalhor, Antonio Guardiani, Yun Meng, Kai Zou, Sergiy Dobrovolskiy, Andreas W. Fognini, Dennis R. Schaart, Dan Dalacu, Philip J. Poole, Michael E. Reimer, Xiaolong Hu, Silvania F. Pereira, Val Zwiller, and Sander N. Dorenbos. Efficient single-photon detection with 7.7 ps time resolution for photon-correlation measurements. *ACS Photonics*, 2020.

[10] J. P. Allmaras and B. A. Korzh, M. D. Shaw, A. G. Kozorezov, and K. K. Berggren. Intrinsic timing jitter and latency in superconducting single photon nanowire detectors. 2018.

[11] Ni Yao, Quan Yao, Xiu-Ping Xie, Yang Liu, Peizhen Xu, Wei Fang, Ming-Yang Zheng, Jingyun Fan, Qiang Zhang, Limin Tong, and Jian-Wei Pan. Optimizing up-conversion single-photon detectors for quantum key distribution. *Optical express*, 2020.

[12] J. Chang, J. W. N. Los, J. O. Tenorio-Pear, N. Noordzij, R. Gourgues, A. Guardiani, J. R. Zichi, S. F. Pereira, H. P. Urbach, V. Zwiller, S. N. Dorenbos, and I. Esmaeil Zadeh. Detecting telecom single photons with 99% system detection efficiency and high time resolution. *APL Photonics*, 2021.

[13] Y. Gisin, G. Ribordy, W. Tittel, and H. Zbinden. Quantum cryptography. reviews of modern physics. pages 145–190, 2002.

[14] C. Zhou, G. Wu, X. Chen, H. Li, and H. Zeng. Quantum key distribution in 50-km optic fibers. *Science China: Physics,Mechanics Astronomy 47:182–188*, 2004.

[15] Mikhail Y. Berezin* and Samuel Achilefu. *Fluorescence Lifetime Measurements and Biological Imaging*. 2010.

[16] Jin Chang, Iman Esmaeil Zadeh, Johannes W. N. Los, Julien Zichi, Andreas Fognini, Monique Gevers, Sander Dorenbos, Silvania F. Pereira, Paul Urbach, and Val Zwiller. Multimode-fiber-coupled superconducting nanowire single-photon detectors with high detection efficiency and time resolution. 2019.

[17] WeiJun Zhang, LiXing You, Hao Li, Jia Huang, ChaoLin Lv, Lu Zhang, XiaoYu Liu, JunJie Wu, Zhen Wang, and XiaoMing Xie. Nbn superconducting nanowire single photon detector with efficiency over 90% at 1550 nm wavelength operational at compact cryocooler temperature. 2017.

[18] J. Bardeen, L. N. Cooper, and J. R. Schrieffer. Theory of superconductivity. pages 1175–1204, 1957.

[19] Francesco Marsili, Faraz Najafi, Eric Dauler, Francesco Bellei, Xiaolong Hu, Maria Csete, Richard J. Molnar, and Karl K. Berggrent. Single-photon detectors based on ultranarrow superconducting nanowires. *Nano Letters*, 2011.

[20] Chandra M Natarajan, Michael G Tanner, and Robert H Hadfield. Superconducting nanowire single-photon detectors: physics and applications. *Superconductor Science and Technology, Volume 25, Number 6*, 2012.

[21] Joel K.W. Yang, Andrew J. Kerman, V. Anant, and E.A. Dauler. Modeling the electrical and thermal response of superconducting nanowire single-photon detectors;modeling the electrical and thermal response of superconducting nanowire single-photon detectors. 2007.

[22] Junjie Wu, Lixing You, Sijing Chen, Hao Li, Yuhao He, Chaolin Lv, Zhen Wang, and Xiaoming Xie. Improving the timing jitter of a superconducting nanowire single-photon detection system. 2017.

Appendix

In this part of the document, the Python code is displayed.

Python

Here, the Python code developed for this project is shown. It is divided into three sections, the first two are libraries used to communicate with the hardware and for the data processing respectively. The last section is the code used to call the aforementioned libraries. The code can also be easily accessed on GitHub, using the following URL:
https://github.com/ianmrosales/SNSPD_efficiencies.

Functions.py

This library is written to communicate with the hardware: the laser, attenuator, power meters and SNSPD driver. Lots of commentaries have already been written inside the functions to clarify the input parameters and algorithms inside the functions.

```
from WebSQControl import WebSQControl
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def current_setter(number_of_detectors, Irange):
    """
    This function sets the current for all detectors in the SNSPD system in one time
    INPUT:
        number_of_detectors: (INT)
        Irange: INT or LIST, specify the value(s) to be used as current (microA)
    """

    if isinstance(Irange, list):
        y = [[val]*number_of_detectors for val in Irange]
    elif isinstance(Irange, np.ndarray):
        y = [[val]*number_of_detectors for val in Irange]
    else:
        y = [Irange]*number_of_detectors
    return y

def start_snspd(N, tcp_ip_address, control_port, counts_port):
    """

    Starting the SNSPD driver and doing a system check.
    Returns present parameters of the system

    INPUT:
        tcp_ip_address (STR)
        control_port (INT)
        counts_port (INT)
```

```
32
33      OUTPUT:
34          ms_time = integration time of the system (INT)
35          bias_current (LIST,INT)
36          trigger (INT)
37          number_of_detectors (INT)
38
39      """
40      websq = WebSQControl(TCP_IP_ADR = tcp_ip_address, CONTROL_PORT =
        ↪  control_port, COUNTS_PORT = counts_port)
41      websq.connect()
42
43      #Acquire number of detectors in the system
44      number_of_detectors =  websq.get_number_of_detectors()
45      print("Your system has " + str(number_of_detectors) + '
        ↪  detectors\n')
46
47      print("Set integration time to 100 ms\n")
48      websq.set_measurement_periode(100)   #time in ms
49
50      print("Enable detectors\n")
51      websq.enable_detectors(True)
52
53
54      # Print out the parameters of the experiment
55      ms_time = websq.get_measurement_periode()
56      bias_current = websq.get_bias_current()
57      trigger = websq.get_trigger_level()
58
59      #Close connection
60      websq.close()
61
62      print("Read back set values")
63      print("====================\n")
64      print("Measurement Periode (ms): \t" + str(ms_time))
65      print("Bias Currents in uA: \t\t" +    str(bias_current))
66      print("Trigger Levels in mV: \t\t" +   str(trigger))
67
68      # N measurements
69      print("Aquire " + str(N) + " counts measurements")
70      print("============================\n")
71      return ms_time, bias_current, trigger, number_of_detectors
72
73  def detected_counts(tcp_ip_address, control_port, counts_port, N,
    ↪  number_of_detectors, Ib,wav):
74      """
75
76      Parameters
77      ----------
78      tcp_ip_address (STR)
79      control_port (INT)
80      counts_port (INT)
81      N = number of measurements to be taken (INT)
82      number_of_detectors
83      Ib = bias current (LIST), for every detector a value
```

```
    wav = INT, used for naming the xlsx file

    Returns
    -------
    A DataFrame containing all counts, An DataFrame containing the averages per detector.

    """
    websq = WebSQControl(TCP_IP_ADR = tcp_ip_address, CONTROL_PORT = control_port, COUNTS_PORT = counts_port)
    websq.connect()

    # start by setting bias current
    websq.set_bias_current(current_in_uA     = Ib)

    ms_time = websq.get_measurement_periode()

    #Aquire N counts measurements
    #Returns an array filled with N numpy arrays each
    #containing as first element a time stamp and then the detector counts in ascending order.
    counts = websq.aquire_cnts(N)
    for i in range(1,len(counts)):
        counts[i]=counts[i]*1/(ms_time*10**(-3))

    # create dataframe out of the measurements
    df = pd.DataFrame()
    for i in range(len(counts)):
        df = df.append(pd.DataFrame(counts[i]).T)

    # some styling to the indices to make it easier to extract data
    headers = ["Channel "+str(1+i) for i in range(number_of_detectors)]
    meas = [i for i in range(1,N+1)]
    headers.insert(0, "Timestamp")

    df.set_axis(headers, axis=1, inplace=True)
    df.set_axis(meas, axis = 0, inplace=True)

    # remove noise in the used detector
    # add more of such lines if more ports are used
    df = df[df["Channel 7"] > 20000]

    # used if this function is used in a loop for different wavelengths
    df.to_excel("counts"+str(wav)+".xlsx")

    # calculate average counts for each column (detector) in the dataframe
    avgs_detect = df.mean(axis=0)
    websq.close()

    return avgs_detect, df

def count_rate(tcp_ip_address, control_port, counts_port, list_Ib, N, number_of_detectors):
```

```
    """
    This function measures the photon count rate for a range of current values values
    per detector in the system at a set wavelength (set manually)

    Parameters
    -------
    list_Ib: a nested list containing different bias currents
    List inside the list contains currents specified for each detector
    xIb: the range of values for Ib used for the plot

    Returns
    -------
    avgscounts: a list of avg photon counts.

    """

    # create an empty list to store the photon counts
    avgscounts=[]
    for i in list_Ib:
        avgs = detected_counts(tcp_ip_address, control_port, counts_port, N, number_of_detectors, i,0)[0].tolist()

    # since the output of the detected_counts() is a list (containing timestamp),
    # the timestamp is removed and all other values are appended to a list
        for j in avgs[1:]:
            avgscounts.append(j)
    return avgscounts

def get_power():
    import time
    """
    This function retrieves the power from the THOR PM100 power meter.
    Works only if one power meter is connected to the system

    OUTPUT:
        p = power (INT)
    """

    from ctypes import c_uint32,byref,create_string_buffer,c_bool,c_char_p,c_int,c_double
    from TLPM import TLPM

    # establish connection
    tlPM = TLPM()
    deviceCount = c_uint32()
    tlPM.findRsrc(byref(deviceCount))

    print("devices found: " + str(deviceCount.value))

    resourceName = create_string_buffer(1024)

    for i in range(0, deviceCount.value):
```

```
183             tlPM.getRsrcName(c_int(i), resourceName)
184             print(c_char_p(resourceName.raw).value)
185             break
186 
187     tlPM.close()
188 
189     # open the power meter and get power value
190     tlPM = TLPM()
191 
192         #set wavelength power meter
193     tlPM.open(resourceName, c_bool(True), c_bool(True))
194 
195     time.sleep(1)
196     power_fluct = np.zeros(10)
197 
198     j=0
199     while j<10:
200 
201         power =  c_double()
202         tlPM.measPower(byref(power))
203 
204         print(power.value)
205 
206         p = power.value
207 
208         power_fluct[j]=p
209         time.sleep(0.3)
210 
211         print(p)
212         j = j+1
213     tlPM.close()
214     pwr = np.average(power_fluct[2:])
215     print("Power =", pwr )
216     return pwr
217 
218 
219 def calibration(waves, number_of_calimeasurements, time_interval):
220     """
221     Perform measurements on two powermeters for a range of
    ↪  wavelengths. Store the output in xlsx files
222     INPUT:
223         waves = range of wavelengths (LIST)
224         number_of_calimeasurements = the number of power measurements
    ↪  to be taken at each wavelength
225         time_interval = specify the amount of time to wait between
    ↪  each measurement
226 
227     OUTPUT:
228         dflaserP1 = DataFrame containing all powers of powermeter 1,
    ↪  sorted per wavelength
229         dflaserP2 = DataFrame containing all powers of powermeter 2,
    ↪  sorted per wavelength
230         laserlistP1 = List containing all powers of powermeter 1,
    ↪  sorted per wavelength
231         laserlistP2 = List containing all powers of powermeter 2,
    ↪  sorted per wavelength
```

```
    """
    global resourceNameP1, resourceNameP2
    import laser
    import time
    import Attenuator
    from TLPM import TLPM
    from ctypes import cdll,c_long, c_ulong,
    ↪  c_uint32,byref,create_string_buffer,c_bool,c_char_p,c_int,c_int16,c_double
    ↪  sizeof, c_voidp

    # connect with the equipment
    l = laser.Laser()
    l.open_port('192.168.1.149','5') # connects
    d = Attenuator.Attenuator()
    d.open_port('192.168.1.148','18')
    d.setAtt(0)
    tlPM = TLPM()
    deviceCount = c_uint32()
    tlPM.findRsrc(byref(deviceCount))

    print("devices found: " + str(deviceCount.value))

    resourceNameP1 = create_string_buffer(1024)
    resourceNameP2 = create_string_buffer(1024)

    tlPM.getRsrcName(c_int(0), resourceNameP1)
    tlPM.getRsrcName(c_int(1), resourceNameP2)

    tlPM.close()

    # fill up the df
    laserlistP1 = []
    laserlistP2 = []

    for wave in waves:
        print(wave)
        l.setWVL(wave)

        d.setWVL(wave)
        print("================================")
        print("The wavelength is now set to:",l.getWVL())

        power_fluctP1 = []
        power_fluctP2 = []

        j = 0

        tlPM = TLPM()

        tlPM.open(resourceNameP1, c_bool(True), c_bool(True))

        waveset =  c_double(wave)
        tlPM.setWavelength(waveset)
        print("P1 values:")
```

```
285
286         while j < number_of_calimeasurements:
287
288             power =  c_double()
289             tlPM.measPower(byref(power))
290
291             print(power.value)
292
293             p = power.value
294
295             power_fluctP1.append(p)
296
297             j = j+1
298             time.sleep(time_interval)
299
300         tlPM.close()
301         laserlistP1.append(power_fluctP1)
302
303         i = 0
304         tlPM = TLPM()
305         tlPM.open(resourceNameP2, c_bool(True), c_bool(True))
306
307         waveset =  c_double(wave)
308         tlPM.setWavelength(waveset)
309         print("P2 values:")
310
311         while i < number_of_calimeasurements:
312
313             power =  c_double()
314             tlPM.measPower(byref(power))
315
316             print(power.value)
317
318             p = power.value
319
320             power_fluctP2.append(p)
321
322             i = i+1
323             time.sleep(time_interval)
324
325
326         tlPM.close()
327
328         laserlistP2.append(power_fluctP2)
329
330     dflaserP1 = pd.DataFrame(laserlistP1).T
331     headers = [i for i in waves]
332     meas = [i for i in range(1,number_of_calimeasurements+1)]
333
334     dflaserP1.set_axis(headers, axis=1, inplace=True)
335     dflaserP1.set_axis(meas, axis = 0, inplace=True)
336     dflaserP1.to_excel('powerfluctP1.xlsx')
337
338
339     dflaserP2 = pd.DataFrame(laserlistP2).T
```

```
340     dflaserP2.set_axis(headers, axis=1, inplace=True)
341     dflaserP2.set_axis(meas, axis = 0, inplace=True)
342     dflaserP2.to_excel('powerfluctP2.xlsx')
343 
344     return dflaserP1, dflaserP2, laserlistP1, laserlistP2
345 
346 def laser_stability(waves, number_of_calimeasurements, time_interval):
347     """
348     This function measures the stability of the laser across a range
    ↪   of wavelengths
349     taking power measurements every specified time interval
350 
351     Parameters
352     ----------
353     waves : LIST
354         LIST CONTAINING LIST OF WAVELENGTH VALUES.
355     times : LIST
356         DESCRIPTION.
357 
358     Returns
359     -------
360     df : DATAFRAME
361         DF CONTAINING THE POWER FLUCTUATIONS OF THE LASER, EACH COLUMN
    ↪   BEING ANOTHER WAVELENGTH
362 
363     """
364     global resourceName
365     import laser
366     import time
367     from TLPM import TLPM
368     from ctypes import cdll,c_long, c_ulong,
        ↪   c_uint32,byref,create_string_buffer,c_bool,c_char_p,c_int,c_int16,c_double
        ↪   sizeof, c_voidp
369     # connect with the equipment
370     l = laser.Laser()
371     l.open_port('192.168.1.149','5') # connects
372     tlPM = TLPM()
373 
374     resourceName = create_string_buffer(1024)
375     deviceCount = c_uint32()
376     tlPM.findRsrc(byref(deviceCount))
377 
378     for i in range(0, deviceCount.value):
379         tlPM.getRsrcName(c_int(i), resourceName)
380 
381     tlPM.close()
382 
383     # fill up the df
384     laserlist = []# pd.DataFrame(columns = waves)
385     for wave in waves:
386         print(wave)
387         l.setWVL(wave)
388         print("================================")
389         print("The wavelength is now set to:",l.getWVL())
390 
```

```
391         power_fluct = []
392 
393         j = 0
394         tlPM = TLPM()
395         tlPM.open(resourceName, c_bool(True), c_bool(True))
396 
397         waveset =  c_double(wave)
398         tlPM.setWavelength(waveset)
399 
400         while j < number_of_calimeasurements:
401 
402             power =  c_double()
403             tlPM.measPower(byref(power))
404 
405             print(power.value)
406 
407             p = power.value
408 
409             power_fluct.append(p)
410 
411             j = j+1
412             time.sleep(time_interval)
413 
414         tlPM.close()
415         laserlist.append(power_fluct)
416 
417     dflaser = pd.DataFrame(laserlist).T
418     headers = [i for i in waves]
419     meas = [i for i in range(1,number_of_calimeasurements+1)]
420     dflaser.set_axis(headers, axis=1, inplace=True)
421     dflaser.set_axis(meas, axis = 0, inplace=True)
422     dflaser.to_excel('powerfluct.xlsx')
423 
424     return dflaser, laserlist
425 
426 def measurements(tcp_ip_address, control_port, counts_port, N,
    ↪ number_of_detectors, Ib, waves, db, laserip,laserchannel):
427 
428     """
429     Measure the counts for a range of wavelengths, output an xlsx file
    ↪ for each wavelength
430     Measure the power in the reference arm during the measurements,
    ↪ output a final xlsx file with the fluctuations
431 
432     INPUT:
433         tcp_ip_address (STR)
434         control_port (INT)
435         counts_port (INT)
436         N = number of counts per measurement
437         number_of_detectors (INT)
438         Ib = list of bias currents
439         waves = list of wavelengths to measure on (LIST)
440         db = a set attenuation level (INT)
441         laserip (STR)
442         laserchannel (INT)
```

```
    """

    from ctypes import c_uint32,byref,create_string_buffer,c_bool,c_char_p,c_int,c_double,c_int16
    from TLPM import TLPM
    import laser
    import Attenuator
    import time

    l = laser.Laser()
    l.open_port(laserip,laserchannel) # connects

    d = Attenuator.Attenuator()
    d.open_port('192.168.1.148','18')

    tlPM = TLPM()
    resourceName1 = create_string_buffer(10240)

    deviceCount = c_uint32()
    tlPM.findRsrc(byref(deviceCount))
    for i in range(0, deviceCount.value):
        tlPM.getRsrcName(c_int(i), resourceName1)

    # create a df each column containing the efficiency of all detectors per wavelength
    df2 = pd.DataFrame(columns = waves)

    pf = pd.DataFrame(columns = waves)

    for wave in waves:

        l.setWVL(wave)
        d.setAtt(db)
        d.setWVL(wave)

        print("================================")
        print("The wavelength is now set to:",l.getWVL())

        j=0
        time.sleep(10) # let it calibrate
        #set wavelength power meter
        tlPM.open(resourceName1, c_bool(True), c_bool(True))
        # set wavelength
        waveset =  c_double(wave)
        tlPM.setWavelength(waveset)

        print(tlPM.setWavelength(waveset))
        print(waveset.value)

        time.sleep(1)
        power_fluct = np.zeros(10)

        while j<10:

            power =  c_double()
```

```
            tlPM.measPower(byref(power))

            print(power.value)

            p = power.value

            power_fluct[j]=p
            time.sleep(0.5)
            print(p)

            j = j+1
        tlPM.close()

        power_fluct2=power_fluct[3:]
        p = np.average(power_fluct2)
        print(p)

        # get detected counts of the SNSPD system, only average counts
        ↪  of N measurements for all detectors
        SNSPD_counts = detected_counts(tcp_ip_address, control_port,
        ↪  counts_port,N, number_of_detectors, Ib, wave)[0].tolist()
        print(SNSPD_counts)

        df2[wave]=SNSPD_counts
        pf[wave] = power_fluct

    # drop first row containing timestamps
    df2 = df2.iloc[1:,:]
    df2.to_excel("measurements.xlsx")

    pf = pf.iloc[1:,:]
    pf.to_excel("powerfluctuationsf.xlsx")
    print(df2)
    print(pf)

    return df2, pf
```

Graphs.py

This library is written to process the data acquired by calling the functions in the functions.py file. Lots of commentaries have already been written inside the functions to clarify the input parameters and algorithms inside the functions.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

def count_rate_plotter(avgscounts, xIb, number_of_detectors):
    """
    Plot the count rate for different currents

    INPUT:
        avgscounts = nested LIST of counts
        xIb = LIST of currents for x-scale
        number_of_detectors (INT)
```

```
    """
    for k in range(0, number_of_detectors):
        newlist = []
        for i in range(k, len(avgscounts), 8):
            newlist.append(avgscounts[i])

        plt.title("Count rate vs I_bias")
        plt.plot(xIb, newlist,label = "Detector "+str(k+1))
        plt.legend()
    plt.show()
    plt.savefig("countrate.png")
    plt.close()

def plot_efficiency(xIb, efficiency, number_of_detectors):
    """
    Plot the efficiency for each detector as a function of a range of
    currents

    INPUT:
        xIb = range of currents for x-scale, LIST
        efficiency =  LIST
        number_of_detectors = INT
    """
    for k in range(0, number_of_detectors):
        newlist = []
        for i in range(k, len(efficiency), 8):
            newlist.append(efficiency[i])

        plt.title("Efficiency vs I_bias")
        plt.plot(xIb, newlist,label = "Detector "+str(k+1))
        plt.legend()
    plt.show()
    plt.savefig("eff.png")
    plt.close()


def wavelength_plot(waves_list, waves,number_of_detectors):
    """
    This function plots the measured efficiency for different
    wavelengths

    INPUT:
        waves_list = LIST containing a range of wavelengths
        waves = LIST containing a range of wavelengths
        number_of_detectors = INT
    """

    for k in range(0, number_of_detectors):
        newlist = []
        for i in range(k, len(waves_list), 8):
            newlist.append(waves_list[i])

        print(newlist)
        print(len(newlist))
        plt.title("Efficiency vs wavelength")
```

```
66          plt.plot(waves, newlist,label = "Detector "+str(k+1))
67          plt.legend()
68
69      plt.show()
70      plt.savefig("wavelength_eff.png")
71      plt.close()
72
73  # functions to plot the power of the laser
74  def power_plotter(df, times):
75
76      """
77      Plot the measured power per wavelength, in separate graph
78
79      INPUT:
80          df = DataFrame containing all power measurements, sorted per
    ↪   wavelength
81          times = LIST of times for x-scale of the plot
82      """
83      # calculate std deviation of each column and determine its
        ↪   fluctuations
84      for header in df:
85          array = df[header].to_numpy()
86          array = [i*10**3 for i in array]
87          # make nice plot power fluctuations
88          plt.plot(times, array, linestyle = "solid", marker = "*",label
            ↪   = str(header)+" nm")
89          plt.xlabel("Time (s)")
90          plt.ylabel("Power (mV)")
91          plt.title("Stability test Power fluctuation")
92          plt.legend()
93          plt.savefig("powerfluct"+str(header)+".png")
94          plt.show()
95          plt.close()
96
97  def totpower_plotter(df, times):
98
99      """
100     Plot the measured power per wavelength, in one figure
101
102     INPUT:
103         df = DataFrame containing all power measurements, sorted per
    ↪   wavelength
104         times = LIST of times for x-scale of the plot
105     """
106
107     # calculate std deviation of each column and determine its
        ↪   fluctuations
108     for header in df:
109         array = df[header].to_numpy()
110         array = [i*10**3 for i in array]
111         # make nice plot power fluctuations
112         plt.plot(times, array, linestyle = "solid", marker = "*",label
            ↪   = str(header)+" nm")
113     plt.xlabel("Time (s)")
114     plt.ylabel("Power (mV)")
```

```
    plt.title("Stability test Power fluctuation")
    plt.legend()
    plt.savefig("powerflucttot.png")
    plt.show()
    plt.close()

def laser_stability_plot(df, waves):
    """
    THIS FUNCTION OUTPUTS A GRAPH SHOWING THE LASER STABILITY FOR ALL
    ↪ WAVELENGTHS

    Parameters
    ----------
    df : dataframe, excel file
        DATAFRAME CONTAINING ALL THE LASER STABILITY DATA.
    waves : array
        ARRAY CONTAINING ALL CORREPONDING WAVELENGTGHS.

    Returns
    -------
    None.

    """
    data = pd.DataFrame()
    averages = []
    deviations = []
    for header in df:
        array = df[int(header)].to_numpy()
        array = [i*10**3 for i in array]

        averages.append(np.average(array))
        deviations.append(np.std(array))

    # make nice plot power fluctuations
    #plt.plot(waves, averages, linestyle = "solid", marker = "*")

    plt.errorbar(waves, averages, yerr = deviations)
    plt.xlabel("wavelengths (nm)")
    plt.ylabel("Power (mW)")
    plt.title("Laser power")
    plt.savefig("powerfluct"+str(header)+".png")
    plt.grid()
    plt.show()
    plt.close()

    perc = [deviations[i]/averages[i] for i in range(len(averages))]

    data["averages"]= averages
    data["std deviations"] = deviations
    data["%"] = perc
    data.index = ([i for i in df])
    data.to_excel("plot_data.xlsx")

    return averages, deviations, perc, data

```

```
169 def stability_plotter(df, waves):
170     """
171     Plot stability of the power for every wavelength
172 
173     INPUT:
174         df = DataFrame containing all power measurements, sorted per
    ↪   wavelength
175         waves = LIST of wavelengths for x-scale of the plot
176     """
177     stability = []
178     for header in df:
179         array = df[header].to_numpy()
180 
181         std_dev = np.std(array)
182         average = np.average(array)
183         stability.append(std_dev/average)
184 
185     # nice plot for stability, how it changes per wavelength
186     plt.plot(waves, stability)
187     plt.title("Stability per wavelength")
188     plt.xlabel("Wavelength (nm)")
189     plt.ylabel("Stability")
190     plt.savefig("stability.png")
191     plt.show()
192     plt.close()
193 
194     return stability
195 
196 def measurements_plotter(counts_meas, laspower, times,
    ↪   number_of_detectors):
197     """
198     Plot the laser power???
199     """
200 
201     # plot laser power during time of measurement
202     plt.plot(times, laspower)
203     plt.xlabel("time (s)")
204     plt.ylabel("power")
205     plt.title("Power course during measurements")
206     plt.savefig("powerduringmeasurement.png")
207     plt.show()
208     plt.close()
209 
210     #probably have to remove column/row for timestamps counts_meas
211     for i in range(number_of_detectors):
212         array = counts_meas.iloc[ : , i+1 ].to_numpy()#correct for
            ↪  timestamp
213         plt.plot(times,array,label= "Detector "+str(i+1))
214     plt.xlabel("time (s)")
215     plt.ylabel("counts")
216     plt.legend()
217     plt.title("counts per detector over time")
218     plt.savefig("countstime.png")
219     plt.show()
220     plt.close()
```

```
221
222 def getratios2(df1, df2):
223     """
224     Calculate the ratios between the measured powers from the two
    ↪  powermeters for each wavelength
225
226     INPUT:
227         df1 = DataFrame 1 for power meter 1, reference arm
228         df2 = DataFrame 2 for power meter 2, measurement arm
229
230     OUTPUT:
231         ratios = LIST containing the ratios between the measured power
    ↪  meters
232         dbratios = LIST containing the dbratios between the measured
    ↪  power meters
233     """
234
235     ratios = []
236     dbratios = []
237
238     ratiosdf = pd.DataFrame()
239     av1 = []
240     av2 = []
241     for header1 in df1:
242
243         avg1 = np.average(df1[header1].to_numpy()[2:]) #first few
            ↪  measurements are bad
244         avg2 = np.average(df2[header1].to_numpy()[2:])
245
246         ratio = np.abs(avg1/avg2)
247
248         dbratio = 10*np.log10(ratio)
249         av1.append(avg1)
250         av2.append(avg2)
251         dbratios.append(dbratio) # error check code
252         ratios.append(ratio)
253     ratiosdf["Average P1"]=av1
254     ratiosdf["Average P2"]=av2
255     ratiosdf["Ratios"]=ratios
256     ratiosdf["dB ratios"] = dbratios
257     ratiosdf.index = ([i for i in df1])
258     ratiosdf.to_excel("ratios.xlsx")
259     return ratios, dbratios
260
261 def ploteffwav(waves, efficiency, i):
262     plt.plot(waves, efficiency*100, marker = ".", linestyle="solid",
        ↪  label = "Measurement "+str(i+1))
263     plt.title("Efficiency vs wavelength")
264     plt.xlabel("Wavelengths (nm)")
265     plt.ylabel("Efficiency (%)")
266     plt.legend()
267     plt.grid()
268     plt.savefig("Efficiency.png")
269
270
```

```
271 def photon_eff(wav, ratios, pwrs, number_of_detectors,
    ↪ detected_counts, j):
272     """
273     This function calculates the efficiency of the system for a range
    ↪ of wavelenghts
274 
275     - The total number of photons is calculated using the given input
    ↪ power pwrs (LIST) of the reference
276     arm, attenuated to get the power input for the SNSPD.
277 
278     - Ratios is a list containing the ratios for the measurement arm
    ↪ and the reference arm, respectively, for all wavelengths
279 
280     - The detected_counts is the list of (avg) counts corresponding
    ↪ with the list of powers, respectively
281 
282     INPUT:
283         wav = wavelength in nanometer (LIST)
284         ratios = ratio between arm and measurement arm for the range
    ↪ of wavelengths ~ 50dB (LIST)
285         pwrs = measured power in reference arm (LIST)
286         number_of_detectors = INT
287         detected_counts = LIST
288         j = number for naming
289 
290     """
291     analysis = pd.DataFrame()
292     p=[]
293     for i in pwrs[1:]:
294         array = pwrs[i].to_numpy()
295         pi = np.average(array)
296         p.append(pi)
297     p = p[1:]
298 
299     totalp = []
300     index = int((wav[0]-1260)/10) # correct for the right domain in
    ↪ the ratios array
301     for i in range(len(p)):
302         totalp.append(p[i]/ratios[i+index])
303 
304     # calculate energy per photon
305     h = 6.62607015*10**(-34)
306     c = 299792458
307     Ewav = [h*c/(i*10**-9) for i in wav]
308 
309     # Calculate the total amount of photons
310     # assumes power meter is before attenuator, hence it reduces
    ↪ signal
311     total_photons = [totalp[i]/Ewav[i] for i in range(len(totalp))]
312 
313     # get the efficiency for every measurement for every detector
314     efficiency = np.zeros(len(detected_counts))
315 
316     for i in range(len(detected_counts)):
317         efficiency[i] = detected_counts[i]/total_photons[i]
```

```
318
319     analysis["System Power"] = totalp
320     analysis["Total Photon count"] = total_photons
321     analysis["Measured Photon count"] =   detected_counts
322     analysis["Efficiency"] = efficiency
323     analysis.index = ([i for i in wav])
324     analysis.to_excel("analysis"+str(wav[0])+"V"+str(j+1)+".xlsx")
325
326     return total_photons, efficiency
```