The effects of disturbances on the Hanbury Brown-Twiss experiment

THESIS

submitted in partial fulfillment of the
requirements for the degree of

BACHELOR OF SCIENCE
in
PHYSICS

Author :	Tim Hogendoorn
Student ID :	s1556452
Supervisor :	Wolfgang Löffler
Second corrector :	John van Noort

Leiden, The Netherlands, June 15, 2021

The effects of disturbances on the Hanbury Brown-Twiss experiment

Tim Hogendoorn

Huygens-Kamerlingh Onnes Laboratory, Leiden University
P.O. Box 9500, 2300 RA Leiden, The Netherlands

June 15, 2021

Abstract

The goal of this research is to find out the effects of certain disturbances on the results of the Hanbury Brown-Twiss (HBT) experiment. The HBT experiment was performed to demonstrate the working principles of the intensity interferometer. In order to model the disturbances a computer model of the HBT experiment was developed in Python. A total of three disturbances were modeled; detector efficiency, detector jitter, and detector dead time, and two types of light were modeled; Coherent and single-photon light. Numerical experiments were performed to measure the effect of the disturbances on the measurements. For detector efficiency and dead time, a clear effect could be observed, but for the detector jitter, the results were dependent on the type of light being used.

Contents

1 Introduction **2**
- 1.1 The goal of this research 2
- 1.2 HBT experiment 2
- 1.3 Additional theory 3
  - 1.3.1 Principle of the HBT experiment and $g^{(2)}$ function 3
  - 1.3.2 Poissonian statistics 4
  - 1.3.3 Photon bunching & antibunching 4

2 Methods **6**
- 2.1 The goal 6
- 2.2 The model 6
  - 2.2.1 Photon generation 6
  - 2.2.2 Beamsplitter 8
  - 2.2.3 Photon detector 8
  - 2.2.4 g2 function 10
- 2.3 Verification of photon generators 11
  - 2.3.1 Coherent light 11
  - 2.3.2 Single-photon light 11
- 2.4 Numerical experiments 12
  - 2.4.1 Control experiments 13
  - 2.4.2 Disturbances 14

3 Results **15**
- 3.1 Detector efficiency 16
  - 3.1.1 Coherent light 16
  - 3.1.2 Single-photon light 17
- 3.2 Detector jitter 19
  - 3.2.1 Coherent light 19

3.2.2 Single-photon light 20
3.3 Dead time 22
3.3.1 Coherent light 22
3.3.2 Single-photon lightx 23

4 Discussion 27
4.1 Summary of results 27
4.2 Points of improvement 28
4.3 Possible future research 28

A Beamsplitter optimisation 31

B Results from all experiments 33
B.1 Efficiency 33
B.1.1 Coherent light 33
B.1.2 Single-photon light 34
B.2 Jitter 36
B.2.1 Coherent light 36
B.2.2 Single-photon light 36
B.3 Dead time 38
B.3.1 Coherent light 38
B.3.2 Single-photon light 38

C Code blocks 40
C.1 Photon generation 40
C.2 Beamsplitter 41
C.3 Click detector 42
C.4 Disturbances 43
C.5 $g^{(2)}$ function 44

Acknowledgements

I would like to thank several people who have helped me in the process of writing this bachelor's thesis. First, I would like to thank my professor, Wolfgang Löffler, who has been immensely helpful during the process of doing my research and during the writing process of this thesis.

Second, I would like to thank my parents for their support and patience during the entire process. I would also like to thank my siblings, Florine and Chris, for their advice during the writing process.

Lastly, I would like to thank my mentors, Kenny and Jelle, for their support and feedback.

Chapter 1

Introduction

1.1 The goal of this research

The goal of this research is to find out the effects of disturbances on the result of the Hanbury Brown-Twiss (HBT) experiment. This question has been divided into several subquestions:

- What disturbances are there on the HBT experiment?
- What are the effects of a non-perfect detector efficiency on the second-order correlation function $g^{(2)}(\tau)$?
- What are the effects of detector jitter on $g^{(2)}(\tau)$?
- What are the effects of detector dead time on $g^{(2)}(\tau)$?

1.2 HBT experiment

Robert Hanbury Brown and Richard Q. Twiss were astronomers with an interest in measuring the diameter of stars. For this purpose they had developed an intensity interferometer, using the 2.5 m telescope at the Mount Wilson Observatory. This experiment proved useful for measuring the diameter of several stars, among which that of Betelgeuse. However, their methods became disputed.[1] In order to address these disputes, they decided to test the principles of their experiment in a laboratory. This experiment is known as the HBT experiment.

The purpose of the HBT experiment is to show that the theory behind the intensity interferometer used by Hanbury Brown and Twiss is sound.

The HBT experiment is an experiment with the purpose of determining the correlation of photons within a photon stream. The correlation of photons is a measure of how bunched up photons are within a photon stream.

The original experiment has been illustrated in Figure 1.1. The photon emitter was a mercury lamp, and the light from the lamp was focussed and filtered such that only light with a wavelength of 435.8 nm fell on the beamsplitter. For the beamsplitter a half-silvered mirror was used, which divided the impinging light into two streams towards a pair of photomultipliers. The outputs of the photomultipliers were then amplified and fed into a correlator, which returned the correlation between the fluctuations in the outputs [2].

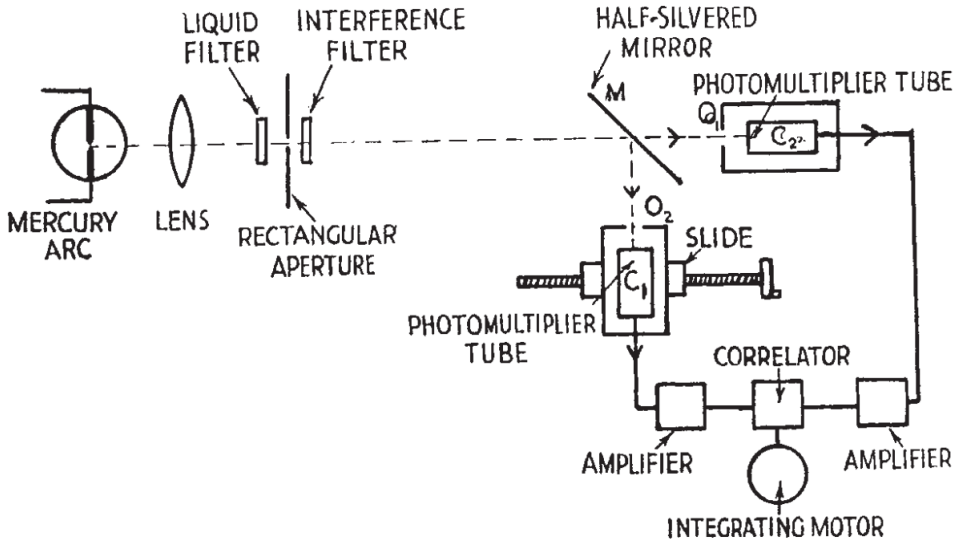

Figure 1.1: *The original experimental setup used for the HBT experiment [2]*

1.3 Additional theory

1.3.1 Principle of the HBT experiment and $g^{(2)}$ function

The basic experimental setup consists of a photon emitter, a 50/50 beamsplitter, and two time-tagged photon detectors. During the experiment, the light from the source is sent to the half-silvered mirror, which functions as a 50/50 beamsplitter. Here, the photon stream gets split into two separate streams. These two streams then impinge on the photon detectors

(C1 and C2 in Fig 1.1), which produce a click whenever a photon falls on them (with a certain probability). The correlator unit then determines the amount of coincidental clicks. The slide which photodetector C1 is attached to can be used to introduce a delay τ in the measurements of the second photodetector. From this the second-order correlation function $g^{(2)}(\tau)$ can be calculated, which is a measure of the probability of two photons falling on both detectors with a τ time interval.

$$g^{(2)}(\tau) = \frac{\langle n_1(t) n_2(t+\tau) \rangle}{\langle n_1(t) \rangle \langle n_2(t+\tau) \rangle} \tag{1.1}$$

Here, n_1 and n_2 are the amount of photons that impinge on the photon detectors, and the angled brackets $\langle ... \rangle$ indicate the time average, which is calculated by integrating over a long time period.

1.3.2 Poissonian statistics

The most well-known kind of classical light is coherent light, which is light which has constant angular frequency ω, phase ϕ, and amplitude $\mathcal{E}_0$. The equation discribing that light is as follows:

$$\mathcal{E}(x,t) = \mathcal{E}_0 \sin(kx - \omega t + \phi) \tag{1.2}$$

Coherent light follows Poissonian photon statistics. This means that the amount of photons within a subsection of the photon stream follows the Poissonian photon distribution:

$$P(n) = \frac{\overline{n}^n}{n!} e^{-\overline{n}} \tag{1.3}$$

Here, n is the amount of photons within that subsection of the photon beam, and $\overline{n}$ is the average amount of photons in a subsection of the same length.

This means that coherent light still has some variation in the amount of photons in each subsection. The standard deviation of the distribution is given by

$$\Delta n = \sqrt{\overline{n}} \tag{1.4}$$

1.3.3 Photon bunching & antibunching

Light can be classified in many different ways. One of them is by what the value of $g^{(2)}(0)$ is for that light. There are three distinct possibilities:

- $g^{(2)}(0) > 1$: Bunched light
- $g^{(2)}(0) = 1$: Coherent light
- $g^{(2)}(0) < 1$: Antibunched light

Coherent light was mentioned before in the previous section. Because coherent light has random time intervals between photons, the probability of detecting a photon τ after another photon detection is the same for all values of τ. Therefore, coherent light must have $g^{(2)}(\tau) = 1$ for all values of τ.

Bunched light is light in which the photons tend to clump up into regions with either more or fewer photons. This means that photons tend to form up in groups or "bunches". As a result of this, the probability of detecting a second photon short after a first photon is higher than detecting a second photon after a longer time. Therefore, it is expected that the value of $g^{(2)}(\tau)$ is larger for small values of τ than for higher values.

All classical light must satisfy the following equations:

$$g^{(2)}(0) \geq 1 \tag{1.5}$$

$$g^{(2)}(0) \geq g^{(2)}(\tau) \tag{1.6}$$

For example, thermal light has $g^{(2)}(0) = 2$, and is therefore bunched.

Antibunched light is light where the gaps between photons are regular, as opposed to having random length. Because of this, the probability of detecting a photon right after another one is low, while the probability of detecting one after a longer time increases with the amount of time. Because of this, the following holds for antibunched light:

$$g^{(2)}(0) < g^{(2)}(\tau) \tag{1.7}$$

$$g^{(2)}(0) < 1 \tag{1.8}$$

This is in violation of the equations 1.5 and 1.6, which apply to classical light. Therefore, photon antibunching is a quantum effect without any classical equivalent [3].

Chapter 2

Methods

2.1 The goal

The purpose of this research is to find out the effects of certain disturbances on the result of the HBT experiment. In order to do this, a model has been constructed (as described in section 2.2). Then, this model has been used to perform several simulations with and without the effects of the disturbances. These simulations are described in section 2.3, and the results of these are described in chapter 3.

2.2 The model

The goal of the model is to simulate an ideal version of the Hanbury-Brown-Twiss experiment, with the ability to add certain disturbances to the simulation that might affect the results. The model is written in Python 3, using Jupyter Notebook as editor.

In this section the various functions in the model are explained. The code used for these functions can be found in Appendix C.

2.2.1 Photon generation

Central to the model is the photon stream. The photon stream is modeled as an array filled with integers. Each index of the array represents the amount of photons emitted within a certain timeframe. An example is provided in Figure 2.1. The amount of time each index represents is adjustable in the code.

```
[3 0 3 1 2 1 0 1 0 2 0 2 3 1 1 1 2 2 1 1]
```

Figure 2.1: *An example of a photon stream as represented in the code*

In the model, there are two functions for generating photon stream arrays: One for coherent light and one for single-photon light. These functions each generate a photon stream array, with the amount of photons chosen according to the probabiliy distribution corresponding to the type of light being generated. The function for coherent light uses the Poisson distribution:

$$P_{coh}(n) = \frac{\overline{n}^n}{n!} e^{-\overline{n}} \tag{2.1}$$

The way single-photon light is generated is a bit more complex. Unfortunately, there is no simple formula for the photon distribution of single-photon light. Therefore, another method is required to generate this type of light. A single-photon emitter is an source that discharges only a single photon per emission event. This is typically done with a two-level quantum system, in which the system is put in its excited state by some mechanism, and once the system decays to its ground state, a single photon is emitted.[4] We can use this to calculate the delay between emission events. The total delay t is the sum of the delay of the ground-excited transition t_{12} and the delay of the excited-ground transition t_{21}. The probability for these is as follows:

$$P_{12}(t) = k_{12} e^{-k_{12} t} \tag{2.2}$$

$$P_{21}(t) = k_{21} e^{-k_{21} t} \tag{2.3}$$

Here k_{12} and k_{21} are the excitation rate and decay rate, respectively.

The way this is implemented in the code is very simple for coherent light, but more complex for single-photon light. The function for coherent light uses the `np.random.poisson` function from the numpy module, which creates an array with values distributed according to the Poisson distribution.

The `sp_stream` function keeps track of whether the emitter is in the excited state or in the ground state. It then picks a delay time according to Eq. 2.2 and Eq. 2.3. It adds these together, skips that amount of time ahead in the photon stream, and adds a single photon at that time. This process is then repeated until the end of the photon stream.

Version of June 15, 2021– Created June 15, 2021 - 12:58

2.2.2 Beamsplitter

The beamsplitter function works to separate an incoming photon stream into two photon streams. This mirrors the functionality of a 50/50 beam splitter used in the physical HBT experiment.

The code works as follows: First, the function takes the incoming photon stream array. Then, an array δ of the same length as the photon stream is created, and filled with random values between 0 and 1, using `np.random.rand`. Then, for every index, an amount of photons in that index are put in the transmission array. The amount of photons is equal to the fraction in δ multiplied by the amount of photons in the photon stream array, rounded to the nearest integer. The remaining photons are put in the reflection array. After this, both arrays are returned. The amount of photons in each array is represented as follows:

$$n_{trans,i} = \delta_i \cdot n_{inc,i} \tag{2.4}$$

$$n_{refl,i} = n_{inc,i} - n_{trans,i} \tag{2.5}$$

Here $n_{trans,i}$ and $n_{refl,i}$ are the i-th elements of the transmission array and the reflection array, respectively. $n_{inc,i}$ is the i-th element of the incoming photon stream array. δ_i is the i-th element of the array with the fraction of photons that gets transmitted.

This code is not completely equivalent to a 50/50 beamsplitter. In order to correctly simulate a beamsplitter, distribution of photons should have been chosen according to the binomial distribution. The decision to use a flat distribution instead of the binomial distribution was made to speed up the beamsplitter function, as the former function is significantly faster than the latter. This does not significantly affect the distribution of photons. A further explanation is given in Appendix A.

2.2.3 Photon detector

The `click_detector` function simulates the operation of a click detector. This detector recieves an incoming photon stream, and when it detects a photon it gives a 'click'.

There are several features that need to be simulated in such a detector. First, there is the detector jitter, which is a random variation of the produced click after a photon absorption event. There is also the fact that the detector has a limited resolution, meaning that there is a limit on how close two photons can be to each other while the detector can still distinguish them.

The detector also has a nonperfect efficiency, meaning that not all photons that fall on the detector are actually detected. Then there is the detector dead time, which means that after the detector detects a photon, the detector cannot detect for a certain amount of time, during which the detector is 'dead'.

Thus, the order of operations is:

1. Jitter
2. Resolution
3. Efficiency
4. Dead time

The code used to simulate the photon detector can be seen in Figure C.4. First, the function takes the incoming photon stream. Then, it applies the jitter to it, the code for which can be seen in Figure C.5. This code works as follows: First, the code takes the photon stream, and the strength of the jitter. Then, for every element of the photon stream array, it moves the contents a random amount, dependent on the intensity of the jitter. This output is then put into a new array, in a new position, appropriate to how far the contents were moved. The method by which the amount of movement of a photon is calculated is by choosing a random amount of time according to the normal distribution, where the intensity of the jitter is represented by the standard deviation of the distribution.

The limited resolution effectively causes the photon stream to be represented in larger time bins. This is achieved by creating an output array of length equal to the original length, divided by the ratio between the resolution and the length of a timebin:

$$T_{new} = T_{old} \cdot \frac{t_{bin}}{t_{res}} \tag{2.6}$$

Here, T_{new} and T_{old} are the lengths of the new and original arrays, respectively. t_{bin} is the length of each time bin, and t_{res} is the resolution of the detector. Each index within the new array represents a timeframe that is $\frac{t_{res}}{t_{bin}}$ times longer than the original timebin. Then, the photons from the old array need to be represented in the new array. This is done by taking the photons within a timeframe from the old array, adding them together, and putting them into the new array.

Next, the efficiency of the detector is simulated. The efficiency of the detector is an indicator of the chance for a detector to actually detect a

photon impinging on it. There are two options to handle this. The first option is a “flat" type of efficiency, where the amount of photons does not affect the detection chance, and it only matters whether a photon falls on the detector or not. For this efficiency type, the following is true:

$$\begin{cases} P_{det} = 0 & n_{phot} = 0 \\ P_{det} = \varepsilon & n_{phot} > 0 \end{cases} \tag{2.7}$$

Here, P_{det} is the detection chance, n_{phot} is the amount of photons that fall on the detector within the timebin, and ε is the efficiency of the detector.

There is also the "exponential" efficiency type, where each photon that falls on the detector increases the detection chance exponentially. For this efficiency type, the following is true:

$$P_{det} = 1 - (1 - \varepsilon)^{n_{phot}} \tag{2.8}$$

When the detection chance has been determined, it is randomly decided (with a P_{det} probability) whether the photon detector has detected a photon that timebin.

After the detection array has been made, the deadtime is applied. In order to simulate the detector deadtime, the code goes through the entire detection array, and if it finds a detection, it sets all detections to False for a number of elements, equivalent to the length of the deadtime. The code used for the deadtime is shown in Figure C.6.

2.2.4 `g2` function

The `g2` function (which measures the probability of two photons falling on both detectors with a τ time interval) calculates the values of $g^{(2)}(\tau)$ given a photon stream and a range of values for τ. It uses the `beamsplitter` function and the `click_detector` function to simulate the setup of the HBT experiment, and then uses the output of those to calculate $g^{(2)}(\tau)$. ($g^{(2)}(\tau)$ is a measure of the probability of two photons falling on both detectors with a τ time interval)

First, the incoming stream is split into two streams using the beamsplitter (`stream1` and `stream2`). Then, the two streams are individually passed into the `click_detector` function, resulting in two arrays filled with Boolean values (`newstream1` and `newstream2`). Then, a range of values for τ is created, for which the values of $g^{(2)}(\tau)$ will be calculated. Then, for every value of τ, the value of Eq. 1.1 is calculated, with the two arrays

Version of June 15, 2021– Created June 15, 2021 - 12:58

`newstream1` and `newstream2` used as $n_1(t)$ and $n_2(t+\tau)$. The approximation of the integration done over a long time with the brackets $\langle ... \rangle$ is done by, for each index, multiplying the value of both arrays, with the second array having its index offset by τ. After the values of $g^{(2)}(\tau)$ have been calculated, both the array with $g^{(2)}(\tau)$ values and the corresponding τ values is returned.

2.3 Verification of photon generators

In this section it is shown that the photon generator functions generate light that is consistent with the theory. This is added because for single-photon light, additional theory had to be developed in order to properly test the functions.

2.3.1 Coherent light

The function for coherent light must follow the photon distribution of Eq. 1.3.

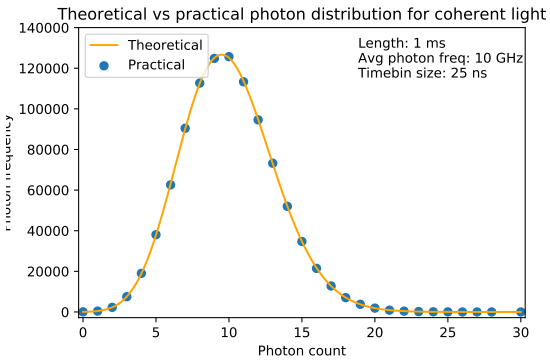

Figure 2.2: *Comparison of theoretical photon distribution and photon distribution generated by the* `coherent_stream` *function*

As can be seen in the graph, the photon distribution of the `coherent_stream` function follows the theoretical distribution perfectly.

2.3.2 Single-photon light

As seen in subsection 2.2.1, there is no simple formula for the photon distribution of single-photon light. However, it is possible to calculate the

expected distribution of delays between photons. It is possible to calculate the probability for the excitation time and the decay time (Eq. 2.2 & 2.3), and this is used to calculate the total emission time $t = t_{12} + t_{21}$.

In order to calculate the probability of the total emission time, we must integrate over all possible combinations of t_{12} and t_{21} that add up to t:

$$P(t) = \int_0^t P_{12}(t_{12})P_{21}(t_{21})dt_{12} \tag{2.9}$$

Filling in 2.2 and 2.3 gives the following:

$$\begin{cases} P(t) = \frac{k_{12}k_{21}}{k_{21}-k_{12}}\left(e^{-k_{12}t} - e^{-k_{21}t}\right) & if k_{12} \neq k_{21} \\ P(t) = k^2 t e^{-kt} & if k_{12} = k_{21} = k \end{cases} \tag{2.10}$$

Plotting this function and comparing it to the delays generated by `sp_stream` gives the following graph:

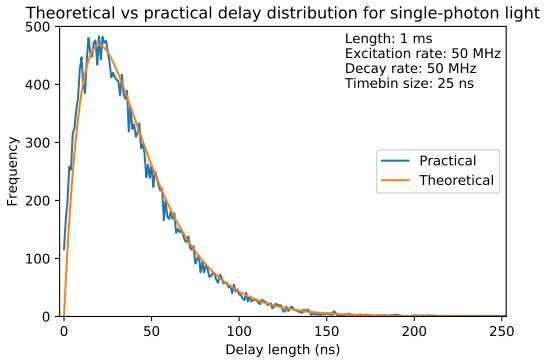

Figure 2.3: *Comparison of theoretical delay distribution and delay distribution generated by the `sp_stream` function*

2.4 Numerical experiments

In order to determine the effects of the disturbances, several numerical experiments have been performed. These experiments each determine the effects of a single type of disturbance. A control experiment has also been performed, to measure the behaviour of $g^{(2)}$ without any disturbances. All of these experiments have been performed for both coherent and single-photon light.

2.4.1 Control experiments

The control experiments have been done for both coherent and single-photon light. One experiment has been done for coherent light, and four experiments have been done for single-photon light.

The control experiment for coherent light has been done with a 10 ms long photon stream, with an average photon frequency of 10 MHz, timebin size of 1 ns, a detector resolution of 25 ns, and a τ range of 10000 ns. The results are shown in Figure 2.4:

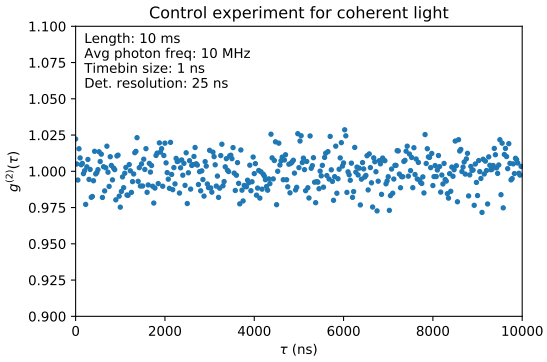

Figure 2.4: *The values for $g^{(2)}(\tau)$ for the control simulation of coherent light*

The graph shows that the value of $g^{(2)}$ stays around 1 for all τ, which is consistent with the theory. There is some noise in the values however, but this does not exceed 3%.

The control experiments for single-photon light have been done for four different parameter combinations. The experiments have been done with an excitation rate of 0.1 GHz and 1 GHz, and a decay rate of 0.1 GHz and 1 GHz, for all combinations of these parameters. The following parameters were the same for all simulations: A timebin size of 1 ns and a detector resolution of 1 ns. The simulation with a k_{12} and k_{21} of 1 has been done with a photon stream length of 1 ms, the other simulations have been done with a stream length of 10 ms. This was done to save on time, due to the time needed for the simulation at such high transition rates. The detector resolution of 1 ns was chosen to better show the curve of $g^{(2)}(\tau)$, as for 25 ns this was not very clear.

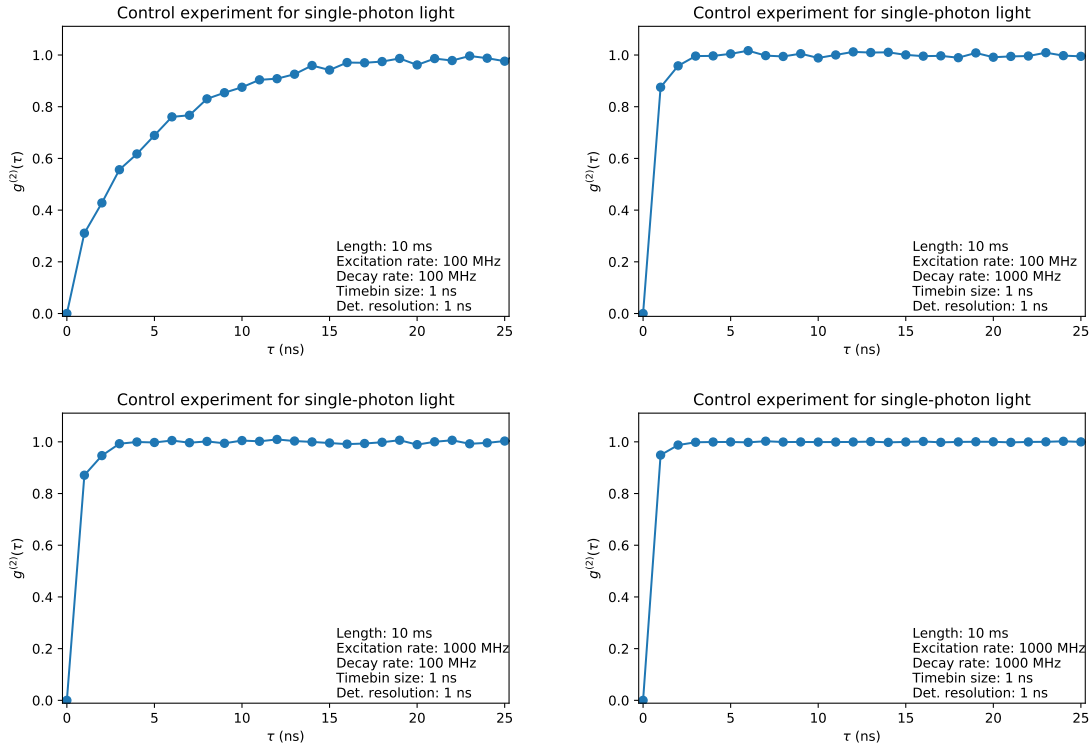

Figure 2.5: *The values for $g^{(2)}(\tau)$ for the control simulations of single-photon light*

The graphs show that the value of $g^{(2)}(\tau)$ starts at 0 for $\tau = 0$, and rises to 1 as τ increases. The rate at which $g^{(2)}(\tau)$ rises increases as the excitation and decay rate become higher. This is consistent with Eq. 1.7 and Eq. 1.8.

2.4.2 Disturbances

After the control simulations, the experiments with the disturbances were performed. The results for these are described in the next chapter. For each of the disturbances, several simulations were run for a range of parameters. These were then compared to the control simulation with the same base parameters.

The first series of simulations was done with non-perfect efficiency, the second was done with the jitter turned on, and the third was done with the dead time turned on.

For the efficiency, the values 75%, 50%, 25%, and 10% were used. For the jitter, the values 1 ns, 4 ns, and 10 ns were used. For the dead time, the values 100 ns and 1000 ns were used.

Chapter 3

Results

The results of the experiments described in the previous chapter are presented here. The experiments can be separated in three categories, each covering two types of light. The categories are the three types of disturbances examined: the detector efficiency, the detector jitter, and the detector dead time.

In order to determine which results are still usable and which results are too impacted to use, acceptance criteria have been made: One for coherent light, and one for single-photon light.

For coherent light, the deviation from the theoretical value of $g^{(2)}(\tau)$ has been used as a marker: The theoretical value is subtracted from all $g^{(2)}$ values from the experiments, leaving only the deviation from the theory.

$$g^{(2)}_{dev}(\tau) = g^{(2)}(\tau) - g^{(2)}_{coh}(\tau) = g^{(2)}(\tau) - 1 \tag{3.1}$$

The standard deviation of these deviation is then calculated. This is done for both the control experiment and the disturbed experiments. The standard deviation of the control experiment $\sigma_{control}$ is then used as a marker of acceptability: If the standard deviation for an experiment is less than twice that of the control experiment, the results of that experiment are still acceptable. Otherwise, the results are too altered to use. The standard deviation for the control experiment is $\sigma_{control} = 0.011$.

For single-photon light, there is no easy theoretical value to compare the results to. Therefore a different method of determining acceptability was chosen. The $g^{(2)}$ values of the control experiment were subtracted from those of the disturbed experiments, leaving the deviation from the control experiment, or the residue:

$$g^{(2)}_{res}(\tau) = g^{(2)}(\tau) - g^{(2)}_{control}(\tau) \tag{3.2}$$

The standard deviation of the residue is then calculated. However, because there is no residue of the control experiment to compare this to, an arbitrary value was chosen instead. A value of 0.02 was chosen as the limit, so if the standard deviation is less than this value, the result is considered acceptable. Otherwise, the results are too altered to use.

3.1 Detector efficiency

These are the results for $g^{(2)}(\tau)$ from simulations with varying detector efficiencies.

3.1.1 Coherent light

The simulations were done with a photon stream of coherent light, with the following properties:

- Array entries: 10000000
- Timebin size: 1 ns
- Total length of stream: 10 ms
- Average photon count per bin: 0.01
- Average photon frequency: 10 MHz
- Detector resolution: 25 ns
- Range of values for τ: 0 ns - 1000 ns, 25 ns interval

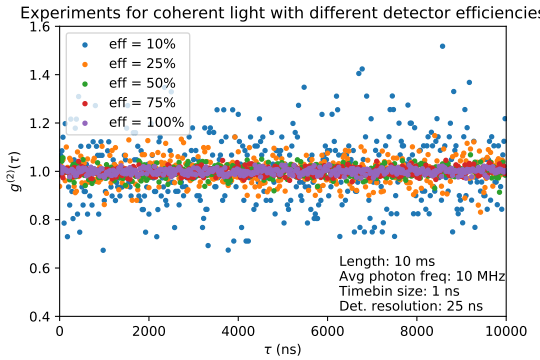

Figure 3.1: *The values of $g^{(2)}(\tau)$ for coherent light, with varying detector efficiencies*

As can be seen, the variations in $g^{(2)}(\tau)$ increase as the efficiency of the detector decreases, with a maximum variation of 0.03 for a 100% efficient detector. This is due to the decrease in efficiency causing there to be less photon detections, which in turn increases the error of the values.

The results are still acceptable for an efficiency of 75% ($\frac{\sigma}{\sigma_{control}} = 1.40$), but are no longer acceptable for an efficiency of 50% ($\frac{\sigma}{\sigma_{control}} = 2.07$). This makes sense, as the total amount of photon detections decreases as the detector efficiency decreases.

3.1.2 Single-photon light

In all simulations with single-photon light, most of the parameters have remained the same. The only parameters that have been altered between simulations are the excitation rate and the decay rate. The constant parameters are as follows:

- Array entries: 10000000
- Timebin size: 1 ns
- Total stream length: 10 ms
- Detector resolution: 1 ns
- Range of value for τ: 0 ns - 1000 ns, interval of 1 ns

The following simulation has been done with an excitation rate of 0.1 GHz and a decay rate of 0.1 GHz.

Version of June 15, 2021– Created June 15, 2021 - 12:58

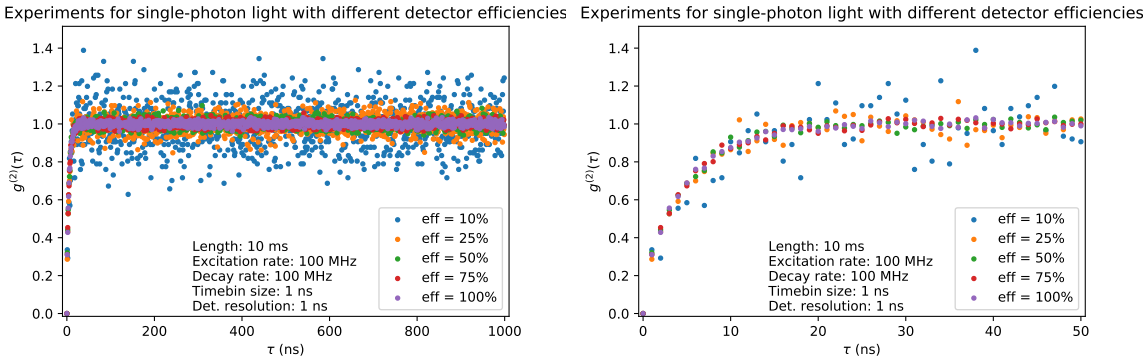

Figure 3.2: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying detector efficiencies. Excitation rate of 0.1 GHz, decay rate of 0.1 GHz*

As can be seen, the general shape of the curve remains the same regardless of detector efficiency, but the dispersion of the values increases as the efficiency of the detector decreases.

For an excitation rate of 0.1 GHz and a decay rate of 0.1 GHz, the results are still acceptable at an efficiency of 75% ($\sigma = 0.019$), but no longer at an efficiency of 50% ($\sigma = 0.025$).

For an excitation rate of 0.1 GHz and a decay rate of 1 GHz, the results are still acceptable at an efficiency of 50% ($\sigma = 0.014$), but no longer at an efficiency of 25% ($\sigma = 0.027$).

For an excitation rate of 1 GHz and a decay rate of 0.1 GHz, the results are still acceptable at an efficiency of 50% ($\sigma = 0.014$), but no longer at an efficiency of 25% ($\sigma = 0.027$).

For an excitation rate of 1 GHz and a decay rate of 1 GHz, the results are still acceptable at an efficiency of 25% ($\sigma = 0.015$), but no longer at an efficiency of 10% ($\sigma = 0.038$).

It can be seen that the increased frequency of photon emissions diminishes the effect of a non-perfect detector efficiency. This is because the direct effect of such an efficiency (decreased amount of photon detections) is at least partially alleviated by the sheer number of photons falling on the detector.

In all simulations with single-photon light it can be seen that the dispersion of values for $g^{(2)}(\tau)$ increases as the efficiency decreases. This is because a decrease in efficiency causes there to be less photon detections, which in turn increases the error of the values.

3.2 Detector jitter

We now turn to the results from the simulations with varying amounts of detector jitter.

3.2.1 Coherent light

The simulations were done with a photon stream of coherent light, with the following properties:

- Array entries: 10000000
- Timebin size: 1 ns
- Total length of stream: 10 ms
- Average photon count per bin: 0.01
- Average photon frequency: 10 MHz
- Detector resolution: 25 ns
- Range of values for τ: 0 ns - 1000 ns, 25 ns interval

The amount of jitter indicated in the legend of the graph is actually the standard deviation of the normal distribution used to generate the amount of jitter.

Version of June 15, 2021– Created June 15, 2021 - 12:58

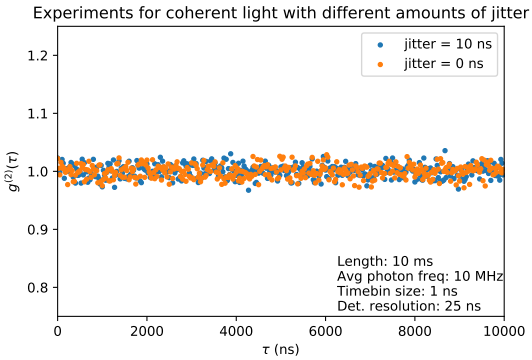

Figure 3.3: *The values of $g^{(2)}(\tau)$ for coherent light, with varying amounts of detector jitter*

As can be seen, this amount of jitter does not seem to have much of an effect on the values of $g^{(2)}(\tau)$. This is also reflected in the standard deviation of the results: $\frac{\sigma}{\sigma_{control}} = 1.001$ for a jitter of 10 ns. This means that the results are acceptable for all examined values of jitter. This is because the jitter adds randomness to the amount of time between photon detections, but for coherent light this is already random, so it has no noticable effect.

3.2.2 Single-photon light

In all simulations with single-photon light, most of the parameters have remained the same. The only parameters that have been altered between simulations are the excitation rate and the decay rate. The constant parameters are as follows

- Array entries: 10000000
- Timebin size: 1 ns
- Total stream length: 10 ms
- Detector resolution: 1 ns
- Range of value for τ: 0 ns - 1000 ns, interval of 1 ns

The following simulation has been done with an excitation rate of 0.1 GHz and a decay rate of 0.1 GHz.

Version of June 15, 2021– Created June 15, 2021 - 12:58

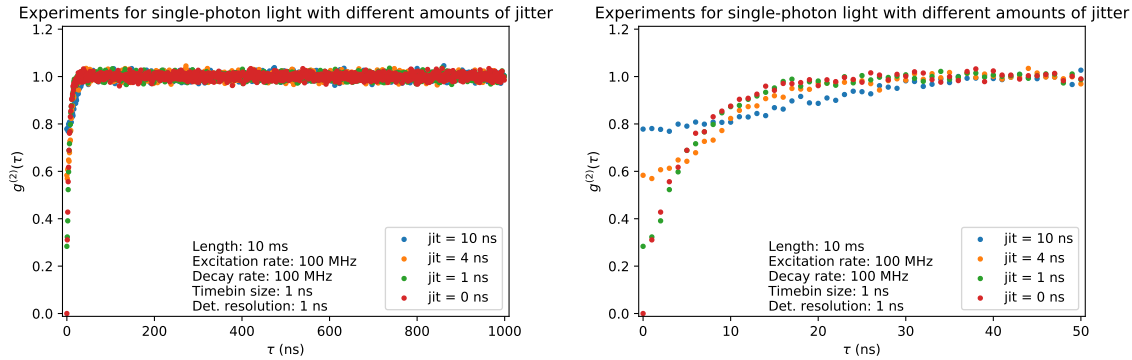

Figure 3.4: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying amounts of detector jitter. Excitation rate of 0.1 GHz, decay rate of 0.1 GHz*

As can be seen, the values of $g^{(2)}(\tau)$ all follow the same curve, but the start and the steepness of the curve are different for the different amounts of jitter. As the amount of jitter increases, $g^{(2)}(0)$ becomes higher, but the width of the dip at $\tau = 0$ decreases.

For an excitation rate of 0.1 GHz and a decay rate of 0.1 GHz, the results are still acceptable at a jitter of 1 ns ($\sigma = 0.019$), but no longer at a jitter of 4 ns ($\sigma = 0.028$).

For an excitation rate of 0.1 GHz and a decay rate of 1 GHz, the results are not acceptable for any of the examined amounts of jitter. ($\sigma = 0.024$ for jitter of 1 ns)

For an excitation rate of 1 GHz and a decay rate of 0.1 GHz, the results are not acceptable for any of the examined amounts of jitter. ($\sigma = 0.024$ for jitter of 1 ns)

For an excitation rate of 1 GHz and a decay rate of 1 GHz, the results are not acceptable for any of the examined amounts of jitter. ($\sigma = 0.025$ for jitter of 1 ns)

Version of June 15, 2021– Created June 15, 2021 - 12:58

Overall, two things have been observed for single-photon light: $g^{(2)}(0)$ is higher when the jitter is higher, and $g^{(2)}(\tau)$ increases more slowly when the jitter is higher. The former can be explained by the fact that jitter rearranges the time of detection for photons, which means that where previously there was no chance of two photons being detected right after each other, there is now a chance of a photon being detected too early, right after the previous photon. Similarly, the “dip" in $g^{(2)}(\tau)$ becomes broader as the jitter increases.

3.3 Dead time

These are the results from the simulations with varying amounts of detector dead time

3.3.1 Coherent light

The simulations were done with a photon stream of coherent light, with the following properties:

- Array entries: 10000000
- Timebin size: 1 ns
- Total length of stream: 10 ms
- Average photon count per bin: 0.01
- Average photon frequency: 10 MHz
- Detector resolution: 25 ns
- Range of values for τ: 0 ns - 1000 ns, 25 ns interval

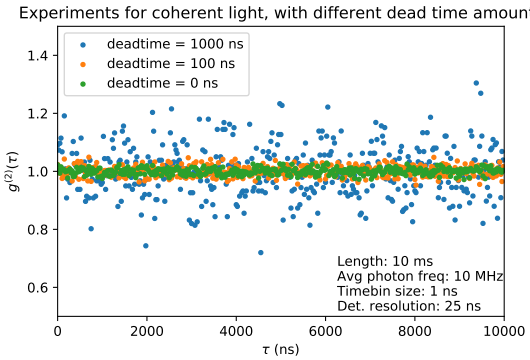

Figure 3.5: *The values of $g^{(2)}(\tau)$ for coherent light, with varying amounts of detector dead time*

Similar to what could be seen in the simulations with non-perfect efficiency, the variation in values of $g^{(2)}(\tau)$ increases as the amount of dead time increases. This is due to the fact that all photon detections during the dead time are missed, reducing the total amount of photon detections, which in turn increases the error of the values.

The results are acceptable for a dead time of 100 ns ($\frac{\sigma}{\sigma_{control}} = 1.60$), but not for a dead time of 1000 ns ($\frac{\sigma}{\sigma_{control}} = 7.88$). This means that the cutoff point for the dead time is likely nearby 100 ns.

3.3.2 Single-photon lightx

In all simulations with single-photon light, most of the parameters have remained the same. The only parameters that have been altered between simulations are the excitation rate and the decay rate. The constant parameters are as follows

- Array entries: 10000000
- Timebin size: 1 ns
- Total stream length: 10 ms
- Detector resolution: 1 ns
- Range of value for τ: 0 ns - 1000 ns, interval of 1 ns

The following simulation has been done with an excitation rate of 0.1 GHz and a decay rate of 0.1 GHz.

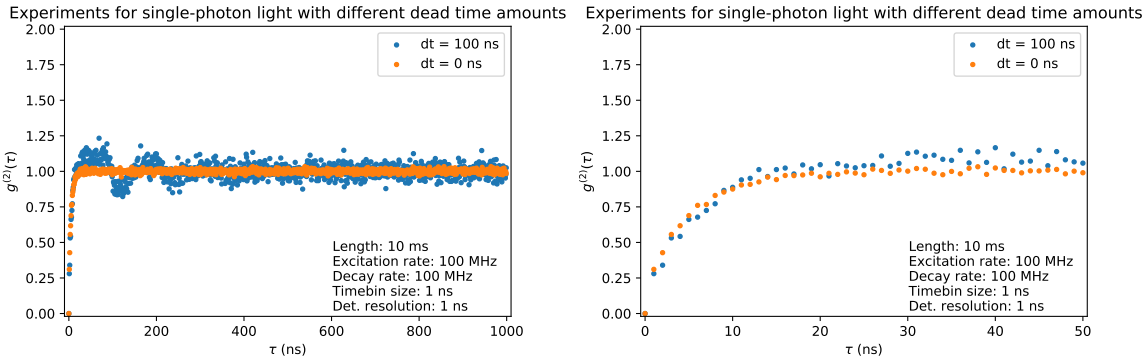

Figure 3.6: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying amounts of detector dead time. Excitation rate of 0.1 GHz, decay rate of 0.1 GHz*

The first panel of Figure 3.6 shows clearly that the dispersion of values for $g^{(2)}(\tau)$ increases as the amount of dead time increases.

Figure 3.7 is from a simulation that has been done with an excitation rate of 1 GHz and a decay rate of 1 GHz. Due to time restraints, this simulation has only been done with 1000000 array entries, adding up to 1 ms total stream length.

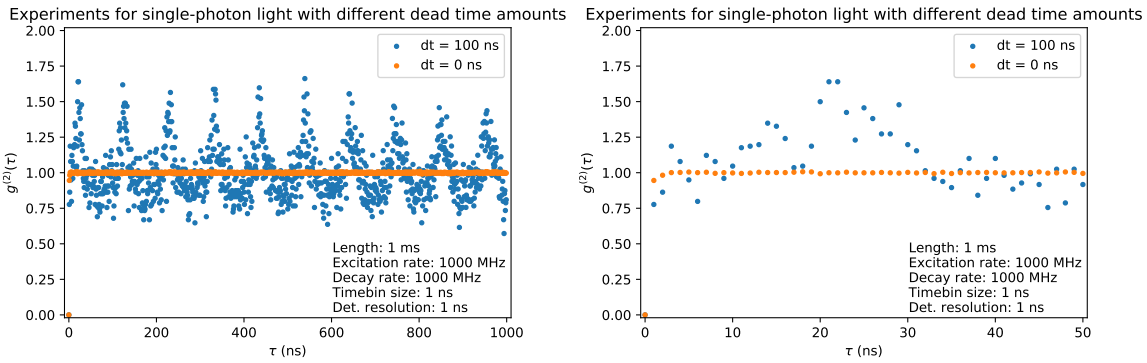

Figure 3.7: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying amounts of detector dead time. Excitation rate of 1 GHz, decay rate of 1 GHz*

It is notable that for a dead time of 100 ns the value of $g^{(2)}(\tau)$ appears to oscillate, with a period equal to the dead time.

The variation in values of $g^{(2)}(\tau)$ increases as the amount of dead time increases. This is due to the fact that all photon detections during the dead time are missed, reducing the total amount of photon detections, which in turn increases the error of the values. Another thing that is noticable is that, especially for the simulations with higher excitation/decay rate, the values of $g^{(2)}(\tau)$ appear to oscillate with a period equal to the length of the dead time. This is because, when the frequency of photons is high, the chance of another photon being detected right after the end of the dead time is high.

For an excitation rate of 0.1 GHz and a decay rate of 0.1 GHz, the results are not acceptable for any of the examined amounts of dead time. ($\sigma = 0.055$ for a dead time of 100 ns)

For an excitation rate of 0.1 GHz and a decay rate of 1 GHz, the results are not acceptable for any of the examined amounts of dead time. ($\sigma = 0.039$ for a dead time of 100 ns)

For an excitation rate of 1 GHz and a decay rate of 0.1 GHz, the results are not acceptable for any of the examined amounts of dead time. ($\sigma = 0.039$ for a dead time of 100 ns)

For an excitation rate of 1 GHz and a decay rate of 1 GHz, the results are not acceptable for any of the examined amounts of dead time. ($\sigma = 0.18$ for a dead time of 100 ns)

Version of June 15, 2021– Created June 15, 2021 - 12:58

This means that the dead time has a considerable effect on the detection of single-photon light.

Chapter 4

Discussion

In this chapter, the results of the experiments are examined and summarised. There is also a recap of the experiments and a review of how they could be improved.

4.1 Summary of results

The main effect of a non-perfect detector efficiency is that as the efficiency decreases, the variation in the values of $g^{(2)}(\tau)$ increases. This seems to be independent of the type of light, as this has been seen in both coherent and single-photon light.

The effect of detector jitter seems to be dependent on the type of light that is being detected; For coherent light the jitter did not have any noticable effect for all values of jitter examined. However, it has a significant effect on single-photon light, as both $g^{(2)}(0)$ and the width of the dip at $\tau = 0$ are affected. As the jitter increases, both $g^{(2)}(0)$ and the width of the dip also increase.

The detector dead time has a similar effect to that of a non-perfect detector efficiency, as it increases the variation in values of $g^{(2)}(\tau)$. Another effect it has on single-photon light with a high photon frequency is that the values of $g^{(2)}(\tau)$ will oscillate with the same period as the dead time, with a gradually decreasing amplitude.

4.2 Points of improvement

A point of improvement for the simulation is the way the photon streams are stored. Currently, photon streams are stored as integers in an array, with each entry representing the amount of photons within a certain timeframe. This has several downsides. One such downside is there being a lot of empty entries, which means that it takes a lot of memory to store a photon stream, and that operations that pass every array entry take a lot of time. Another downside is that the accuracy of the stream is limited by the size of the timeframe each entry represents. Increasing the accuracy of the stream linearly increases the size of the array, which exacerbates the issues mentioned previously.

This could have been prevented by instead of using an array in which every entry represents a timeframe, by using an array where every entry represents a photon, and the stored value is the time at which the photon was emitted. This was not implemented because this would mean that the entire code for the simulations would have to be overhauled, which would have required a massive amount of time and effort.

Another point of improvement is that the model could have been recreated in a physical setup, in order to test its validity. However, due to the Covid-19 pandemic, lab research was not possible.

One of the ways the simulation might be improved is by optimising it so that the simulations run faster, which in turn allows for longer simulations in the same amount of time. One of the ways this could be achieved is by optimising the process of calculating the photon correlations. The current method of calculating this is by using brute force, calculating correlations for each individual element. There are methods for more efficiently calculating autocorrelations, such as using the Wiener-Khinchin theorem, which reduces the order from n^2 to $n\log(n)$. However, these only work for autocorrelations, where both inputs are the same, which is not the case for this simulation. The goal would then be to modify one of these methods in such a way that it would allow for it to work with differing inputs.

4.3 Possible future research

There are many possibilities to continue this research further, or into other areas.

One of the possibilities for expanding this research is to use different types of light in the simulations. Early in the research process there was an attempt to make a photon generator for thermal light, but the complexity

required for this turned out to be too high. Initially, the idea was to simulate thermal light by simply using the Bose-Einstein formula (Eq. 4.1) for the particle distribution, with the thought that that would be enough to properly simulate it, but this seemed to be too simple.

$$P_{BE}(n) = \frac{1}{\overline{n}+1}\left(\frac{\overline{n}}{\overline{n}+1}\right)^{n} \tag{4.1}$$

However, in 2018, several German scientists have done research into properly simulating light emitted by a thermal light source [5]. These methods could be used to simulate thermal light in our own model, in order to analyse the effects of the disturbances on thermal light.

In the current research, only three types of disturbances have been examined. The research question could be further answered if more types of disturbances were examined, such as detector pile-up, or the effects of a finite coherence time. One could also focus more on disturbances that involve the photon source, such as the “blinking" of a single-photon source.

Another research possibility is to determine the effects of multiple disturbances at the same time. This will provide a more realistic simulation of the setup, and may give insight in ways to counteract these disturbances.

Another possibility is to examine one type of disturbance more in depth, in order to more accurately simulate it. This could help with finding ways of more effectively mitigating or eliminating it.

Another possible application is to use the difference between the theoretical photon distribution and the observed photon distribution to determine the amount of detector jitter. One way this could be done is by examining how the observed delay distribution deviates from Eq. 2.10. This can be useful to determine how reliable results are when using a physical setup.

Bibliography

[1] ERIC BRANNEN and H. I. S. FERGUSON. "The Question of Correlation between Photons in Coherent Light Rays". In: *Nature* 178.4531 (Sept. 1956), pp. 481–482. DOI: 10.1038/178481a0. URL: https://doi.org/10.1038%2F178481a0.

[2] R. HANBURY BROWN and R. Q. TWISS. "Correlation between Photons in two Coherent Beams of Light". In: *Nature* 177.4497 (Jan. 1956), pp. 27–29. DOI: 10.1038/177027a0. URL: https://doi.org/10.1038%2F177027a0.

[3] Mark (Anthony Mark) Fox. *Quantum optics an introduction*. eng. Oxford master series in physics ; 15. Oxford ; New York: Oxford University Press, 2006. ISBN: 1-280-96505-3.

[4] Christian Kurtsiefer et al. "Stable Solid-State Source of Single Photons". In: *Phys. Rev. Lett.* 85 (2 July 2000), pp. 290–293. DOI: 10.1103/PhysRevLett.85.290. URL: https://link.aps.org/doi/10.1103/PhysRevLett.85.290.

[5] Raimund Schneider et al. "Simulating the photon stream of a real thermal light source". In: *Applied Optics* 57.24 (Aug. 2018), p. 7076. DOI: 10.1364/ao.57.007076. URL: https://doi.org/10.1364%2Fao.57.007076.

Appendix A

Beamsplitter optimisation

As mentioned in chapter 2, the function for the beamsplitter should use the binomial distribution for a completely correct simulation, but uses a flat distribution instead. This is a valid choice, as for small amounts of photons, the binomial distribution and the flat random distribution are mostly equivalent. This is shown in Figure A.1.

The first and third graphs show a comparison of the frequency of different photon counts, while the second and fourth graphs show the ratio between the frequency of the photon counts. The ratio is calculated as follows:

$$\eta = \frac{n_{rand}}{n_{binomial}} \tag{A.1}$$

As can be seen, the distribution of photons is very similar. The distribution only starts to diverge when the mean photon count exceeds 1 per time bin. The ratio between the frequency of different photon counts is also mostly the same for low photon counts, but starts to increase for higher photon counts. This is expected . However, this is not a significant issue, as the actual number of times this occurs is very low.

Version of June 15, 2021– Created June 15, 2021 - 12:58

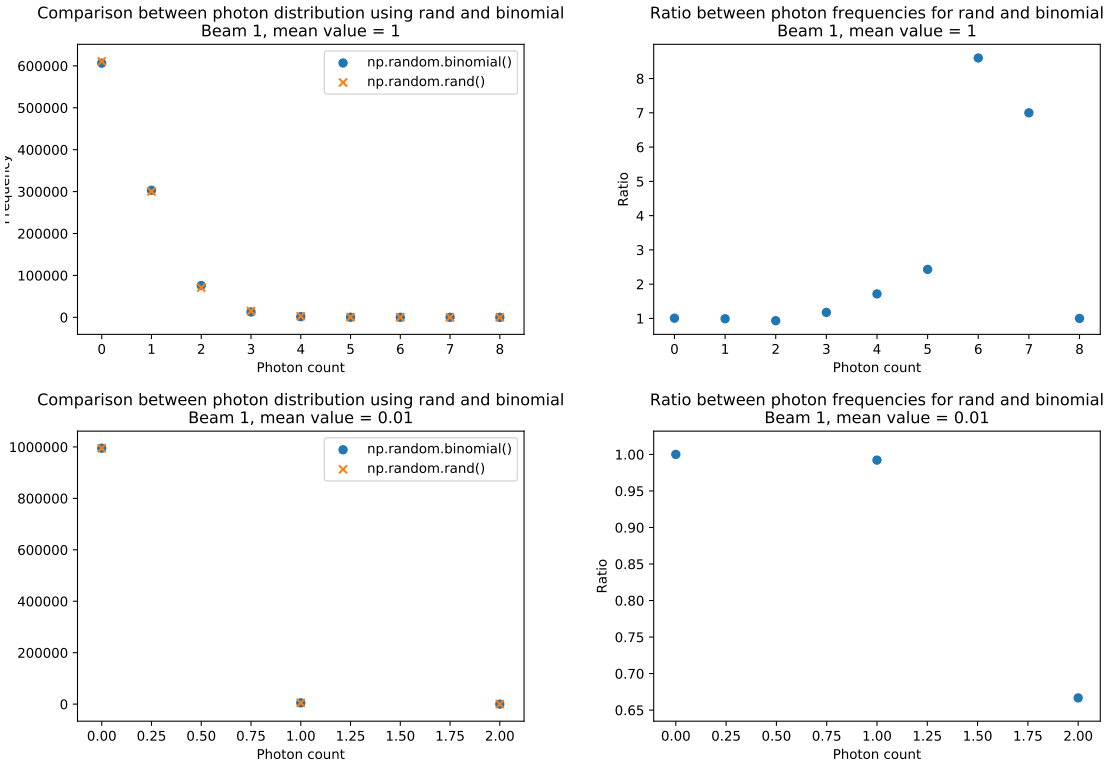

Figure A.1: *A comparison between the photon distribution using the binomial distribution and using a flat distribution*

Appendix B

Results from all experiments

B.1 Efficiency

B.1.1 Coherent light

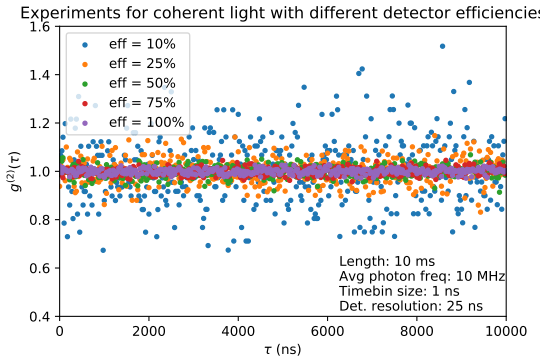

Figure B.1: *The values of $g^{(2)}(\tau)$ for coherent light, with varying detector efficiencies*

B.1.2 Single-photon light

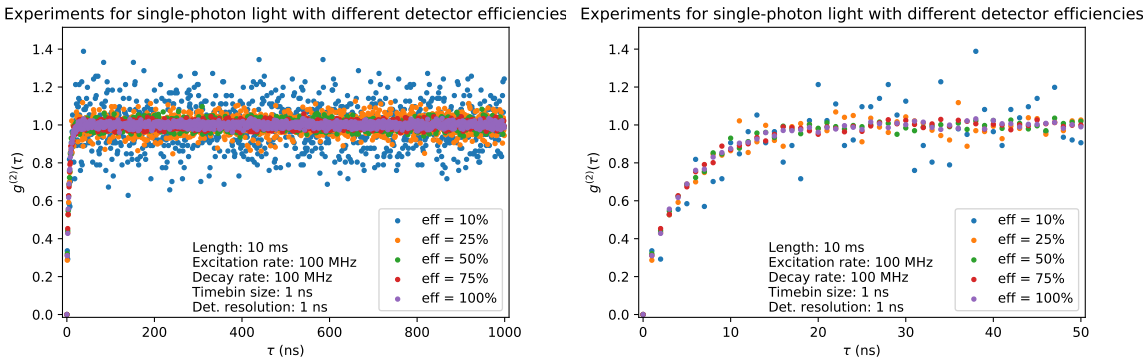

Figure B.2: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying detector efficiencies. Excitation rate of 0.1 GHz, decay rate of 0.1 GHz*

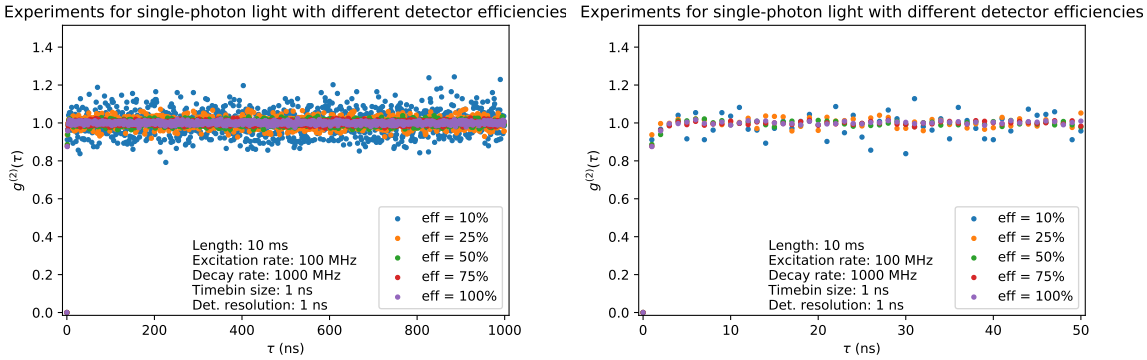

Figure B.3: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying detector efficiencies. Excitation rate of 0.1 GHz, decay rate of 1 GHz*

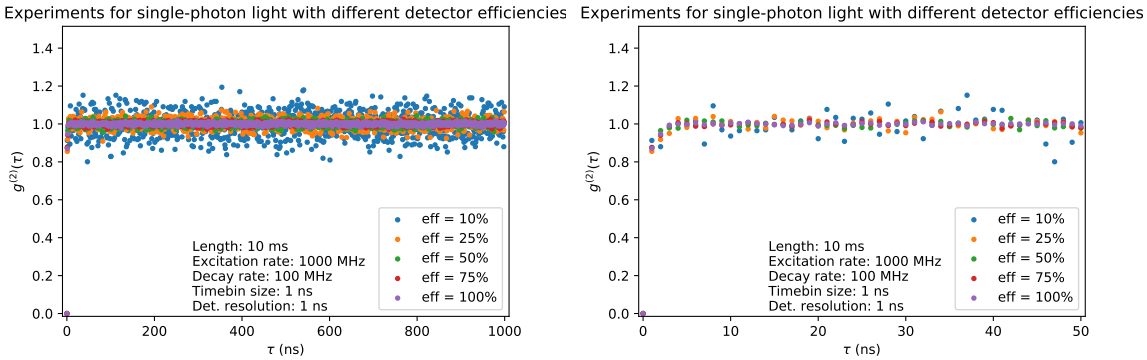

Figure B.4: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying detector efficiencies. Excitation rate of 1 GHz, decay rate of 0.1 GHz*

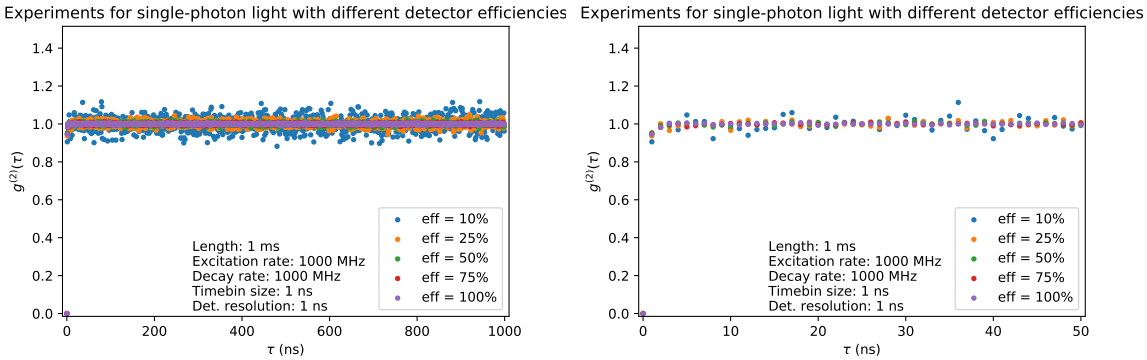

Figure B.5: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying detector efficiencies. Excitation rate of 1 GHz, decay rate of 1 GHz*

B.2 Jitter

B.2.1 Coherent light

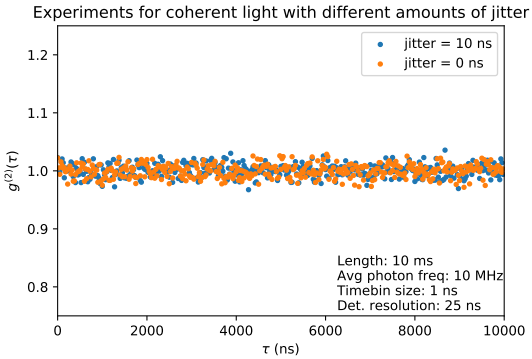

Figure B.6: *The values of $g^{(2)}(\tau)$ for coherent light, with varying amounts of detector jitter*

B.2.2 Single-photon light

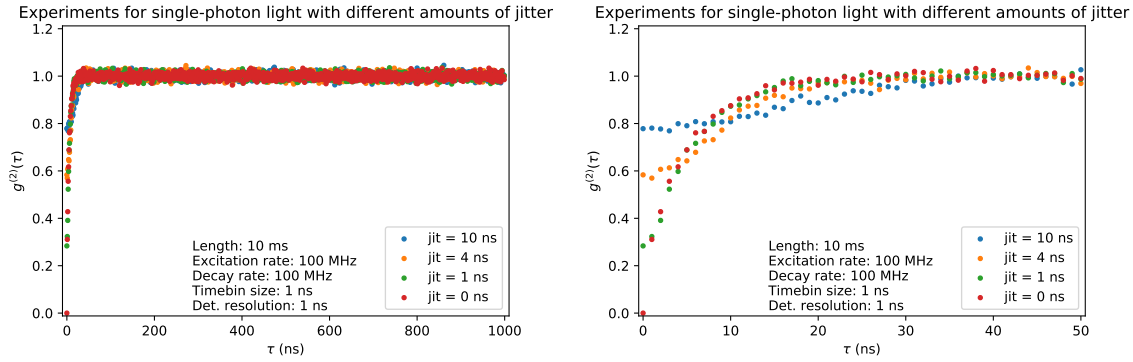

Figure B.7: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying amounts of detector jitter. Excitation rate of 0.1 GHz, decay rate of 0.1 GHz*

Version of June 15, 2021– Created June 15, 2021 - 12:58

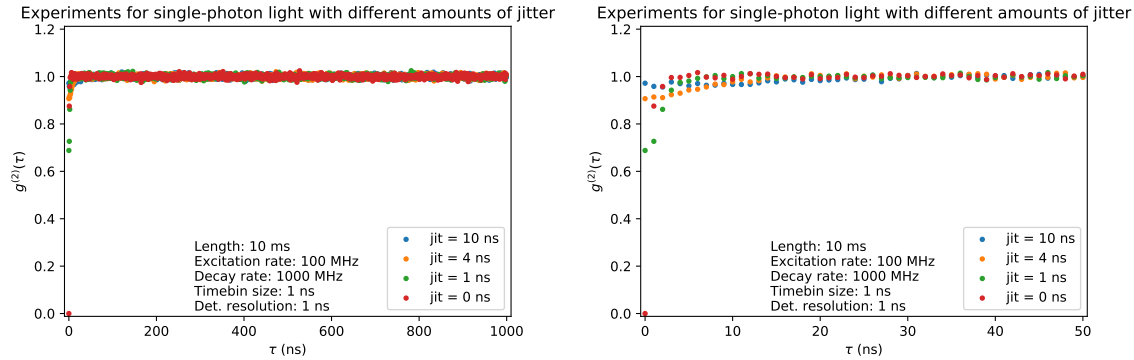

Figure B.8: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying amounts of detector jitter. Excitation rate of 0.1 GHz, decay rate of 1 GHz*

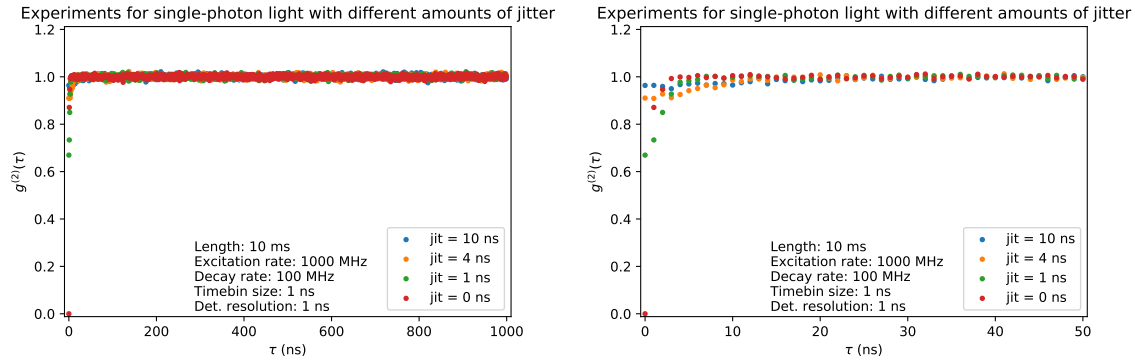

Figure B.9: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying amounts of detector jitter. Excitation rate of 1 GHz, decay rate of 0.1 GHz*

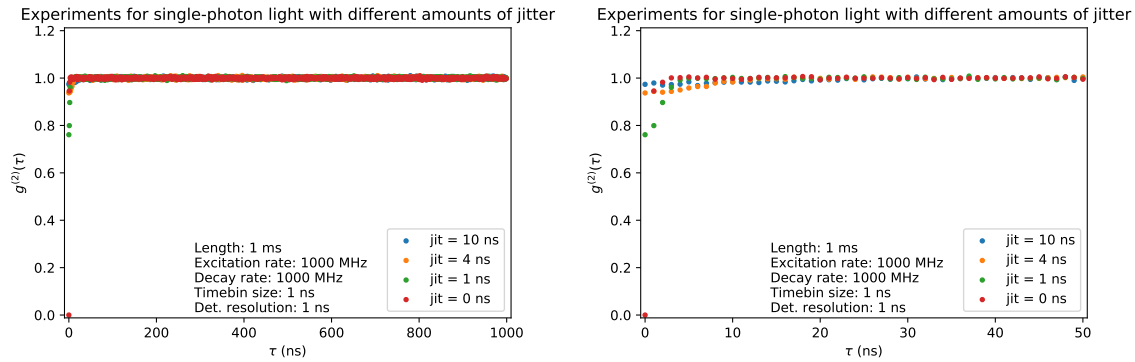

Figure B.10: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying amounts of jitter. Excitation rate of 1 GHz, decay rate of 1 GHz*

B.3 Dead time

B.3.1 Coherent light

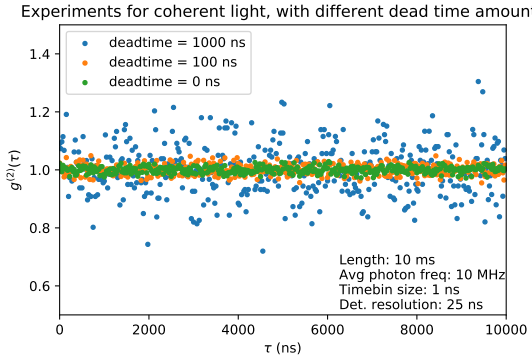

Figure B.11: *The values of $g^{(2)}(\tau)$ for coherent light, with varying amounts of detector dead time*

B.3.2 Single-photon light

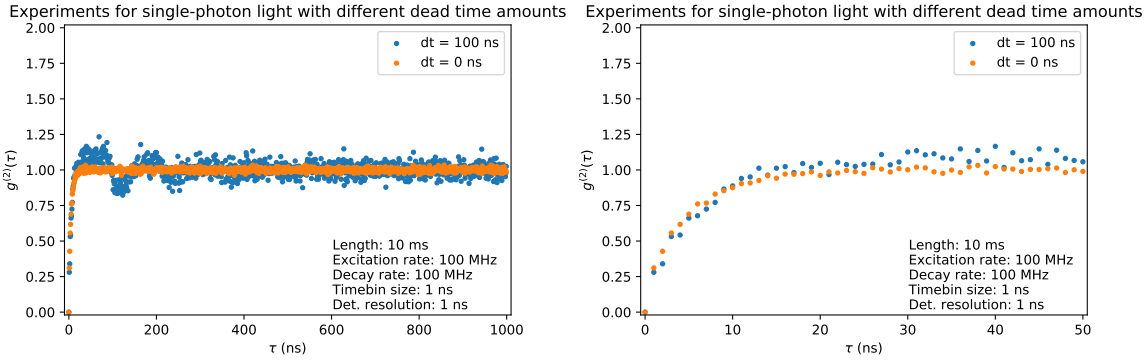

Figure B.12: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying amounts of detector dead time. Excitation rate of 0.1 GHz, decay rate of 0.1 GHz*

Version of June 15, 2021– Created June 15, 2021 - 12:58

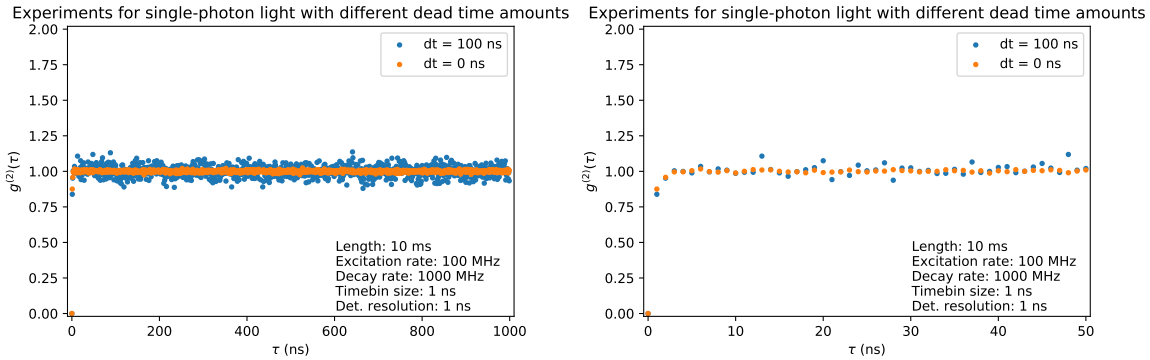

Figure B.13: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying amounts of detector dead time. Excitation rate of 0.1 GHz, decay rate of 1 GHz*

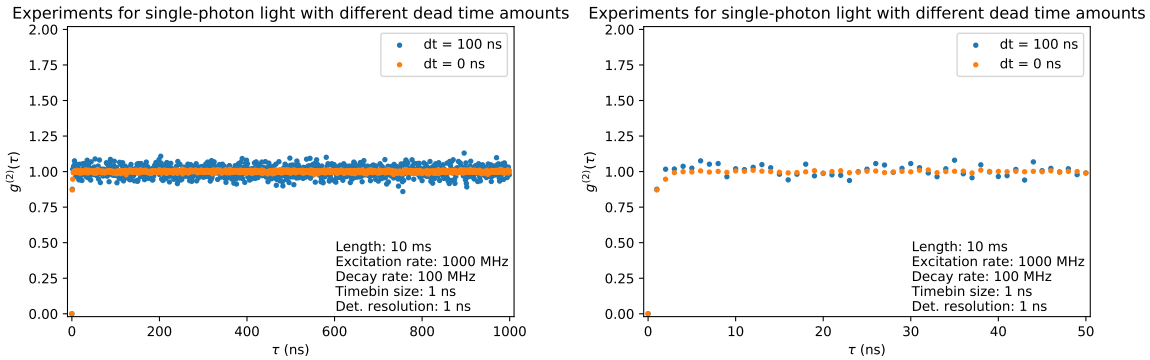

Figure B.14: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying amounts of detector dead time. Excitation rate of 1 GHz, decay rate of 0.1 GHz*

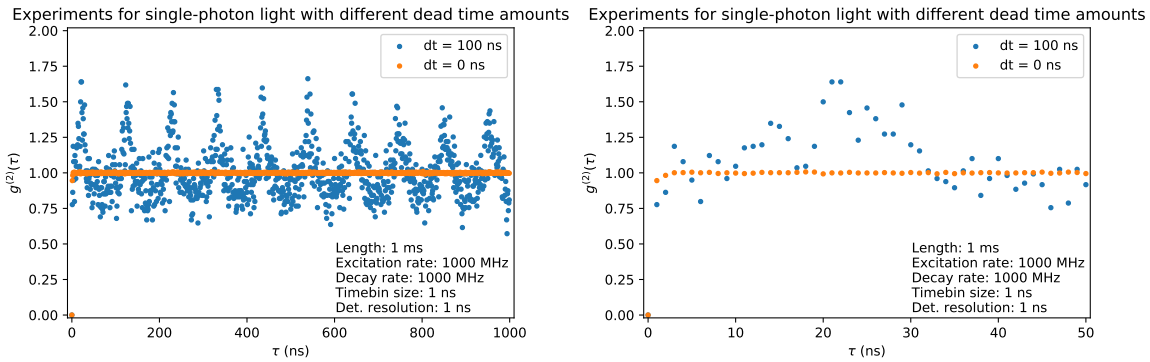

Figure B.15: *The values of $g^{(2)}(\tau)$ for single-photon light, with varying amounts of detector dead time. Excitation rate of 1 GHz, decay rate of 1 GHz*

Appendix C

Code blocks

C.1 Photon generation

```
1  #coherent light
2  def coherent_stream(lam, length):
3      return np.random.poisson(lam, length)
```

Figure C.1: *The code used to generate streams of coherent light*

```
#single-photon light
def sp_stream(k12, k21, length):
    #define variables
    timerange = np.arange(0, length, 1)
    dist_k12 = k12 * np.exp(-k12 * timerange)
    dist_k21 = k21 * np.exp(-k21 * timerange)
    excited = True

    #create output stream
    stream = np.zeros(length, dtype=bool)

    #normalise probability distribution
    dist_k12 = dist_k12 / sum(dist_k12)
    dist_k21 = dist_k21 / sum(dist_k21)

    #prepare arrays with random values
    k12_list = np.random.choice(timerange, size=length, p=dist_k12)
    k12_index = 0
    k21_list = np.random.choice(timerange, size=length, p=dist_k21)
    k21_index = 0

    #set time elapsed to 0
    t = 0

    #during the timeframe of the photon stream
    while(t < length):
        #if the emitter is in an excited state
        if(excited):
            #get the length of the delay before decay (and emission)
            #and skip that much time ahead
            t += k21_list[k21_index]
            k21_index += 1          #increment index

            #check if t is still within time limits
            if(t < length):
                stream[t] = True    #emitter emits photon
                excited = False     #emitter decays to non-excited state

        #if the emitter is not in an excited state
        else:
            #get length of the delay before excitation
            #and skip that much time ahead
            t += k12_list[k12_index]
            k12_index += 1          #increment index

            #emitter is now excited
            excited = True

    return stream
```

Figure C.2: *The code used to generate streams of single-photon light*

C.2 Beamsplitter

```
def beamsplitter(stream):
    length = len(stream)

    trans_array = np.random.rand(length)

    stream_trans = np.around(stream * trans_array).astype(int)
    stream_refl = stream - stream_trans

    return stream_trans, stream_refl
```

Figure C.3: *The code used to split a photon stream into two photon streams*

Version of June 15, 2021– Created June 15, 2021 - 12:58

C.3 Click detector

```
#create function that simulates a click detector detecting a photon stream
def click_detector(stream, resolution, efficiency=1, efftype="flat", jit=0, deadtimelength = 0,
     binlength = 25):
    length = len(stream)
    newlength = int(np.ceil(length/resolution))

    #apply jitter to stream
    stream = jitter(stream, jit, binlength)

    #make output array (filled with False's)
    newstream = np.zeros(newlength, dtype=bool)

    #go through entire new array
    for i in np.arange(0, newlength, 1):
        #count how many photons fall on the detector
        photcount = np.sum(stream[resolution*i:resolution*(i+1)])

        #if photons have fallen on the detector
        if(photcount > 0):
            #determine detection chance
            #if the detector efficiency is not influenced by the amount of photons that fall on it
            if(efftype == "flat"):
                detchance = efficiency

            #if the detector efficiency increases exponentially with every photon
            elif(efftype == "exponential"):
                detchance = 1 - (1 - efficiency) ** photcount

            #if the wrong value for efftype has been given
            else:
                detchance = 0

            if(detchance > np.random.rand()):
                newstream[i] = True


    #retroactively apply dead time
    newstream = deadtime(newstream, int(deadtimelength/resolution), binlength)

    return(newstream)
```

Figure C.4: *The code used to simulate a click detector*

C.4 Disturbances

```
def jitter(stream, jitter, binlength):
    length = len(stream)
    newstream = np.zeros(length, dtype=int)

    #for each element of the stream
    for i in range(length):
        photcount = stream[i]

        #if there is a photon in the current index
        if(photcount > 0):
            #randomly determine the position of the detector
            pos_det = np.random.normal(0, jitter)

            #determine how much earlier the photons are detected
            #for every photon
            for j in range(photcount):
                #randomly determine where in the timebin the photon is
                pos = np.random.rand() * binlength

                #calculate how much the index changes due to jitter
                indexshift = int(np.floor((pos + pos_det)/binlength))

                #put photons in place in new array
                if(i + indexshift >= 0 and i + indexshift < length):
                    newstream[i + indexshift] += 1

    return newstream
```

Figure C.5: *The code used to simulate detector jitter*

```
#define detector dead time function
#deadtimelength and timebin should be the same unit
def deadtime(stream, deadtimelength, timebin):
    #define variables
    length = len(stream)
    deadtimeleft = 0
    newstream = np.zeros(length, dtype=int)

    #go through entire array
    for i in np.arange(0, length, 1):
        #check if there is dead time left
        if(deadtimeleft > 0):
            #detector doesn't detect anything
            newstream[i] = 0

            #if the deadtime doesn't expire within the current timebin
            if(deadtimeleft >= timebin):
                #deadtimeleft is reduced by timebin length
                deadtimeleft = deadtimeleft - timebin
            #if the deadtime expires within the current timebin
            else:
                #deadtimeleft is set to zero
                deadtimeleft = 0

        #if there is no dead time left
        #check if there is a photon detection
        elif(stream[i] != 0):
            #detector detects photons
            newstream[i] = stream[i]
            #deadtime countdown starts
            deadtimeleft = deadtimelength

    return newstream
```

Figure C.6: *The code used to simulate the detector dead time*

Version of June 15, 2021– Created June 15, 2021 - 12:58

C.5 $g^{(2)}$ function

```
1  global TIMEBIN_RESOLUTION
2  TIMEBIN_RESOLUTION = 25
3
4  #optimise g2 function
5  def g2(stream, tau_max, efficiency=1, jit=0, deadtimelength=0, resolution=25):
6      length = len(stream)
7
8      #split the stream
9      stream1, stream2 = beamsplitter(stream)
10
11     #detect streams
12     newstream1 = click_detector(stream1, resolution=resolution, efficiency=efficiency, efftype="flat
           ", jit=jit, deadtimelength=deadtimelength, binlength=TIMEBIN_RESOLUTION)
13     newstream2 = click_detector(stream2, resolution=resolution, efficiency=efficiency, efftype="flat
           ", jit=jit, deadtimelength=deadtimelength, binlength=TIMEBIN_RESOLUTION)
14
15     #create tau range over entire stream length
16     tau = np.arange(0, tau_max/resolution, 1, dtype=int)
17
18     #create array for g2 values
19     g2_array = np.zeros(len(tau))
20
21     #calculate g2 for every value for tau
22     newlength = len(newstream1)
23     for i in tau:
24         g2_array[i] = np.average(newstream1[0:newlength-i] * newstream2[i:newlength])/(np.average(
               newstream1[0:newlength-i]) * np.average(newstream2[i:newlength]))
25
26     return g2_array, resolution*tau
```

Figure C.7: *The code used to calculate the values for $g^{(2)}(\tau)$*

Version of June 15, 2021– Created June 15, 2021 - 12:58