Optical Investigation of Suspended Single Wall Carbon Nanotubes

by

Dávid Lakatos

A thesis submitted in partial fulfillment for the degree of Bachelor of Science

in the

Budapest University of Technology and Economics
Department of Broadband Infocommunications and Electromagnetic Theory
Group of Electromagnetic Theory
and
Delft University of Technology
Department of Nanosciences
Quantum Transport Group

December 2009

Declaration of Authorship

I, David Lakatos, declare that this thesis titled, 'Optical investigation of suspended carbon nanotubes' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed: ______________________________

Date: ______________________________

Contents

Declaration of Authorship **i**

List of Figures **iii**

Abbreviations **iv**

1 Introduction **1**

2 Background **3**
- 2.1 Single wall carbon nanotubes . . . 3
  - 2.1.1 Atomic structure . . . 3
  - 2.1.2 Electronic structure . . . 6
  - 2.1.3 Optical properties . . . 10
- 2.2 Synthesis . . . 12
  - 2.2.1 Sample preparation methods . . . 12
  - 2.2.2 Samples used in the experiments . . . 13

3 Experimental design **16**
- 3.1 Equipment . . . 16
  - 3.1.1 Confocal microscope setup . . . 16
  - 3.1.2 Spectrometer . . . 18
  - 3.1.3 Lasers . . . 21
  - 3.1.4 Motors . . . 21
  - 3.1.5 Power meter . . . 21
- 3.2 Description of the measurement procedures . . . 22
- 3.3 PLE mapping automatization . . . 22
  - 3.3.1 Manual search and sample mapping . . . 23

4 Results **25**
- 4.1 Locating SWCNTs . . . 25
- 4.2 The aging effect of SWCNTs . . . 29
- 4.3 Sample mapping . . . 31
- 4.4 PLE mapping . . . 31

5 Conclusion and Outlook **35**
- 5.1 Conclusion and outlook . . . 35

A Program codes **37**
A.1 RT_Setup.m . . . 37
A.2 getbeam.m . . . 58
A.3 plemap.m . . . 64

B Graphical User Interfaces **77**

C Table for assigning chiral indices **81**

Bibliography **83**

Acknowledgements **86**

List of Figures

2.1 The three types of tubes: zigzag, armchair and chiral. 5
2.2 The chiral and translation vectors of CNTs 5
2.3 Unit vectors in real (a) and reciprocal space (b) 7
2.4 The periodic boundary condition using the ZF approximation 8
2.5 Reciprocal lattice of graphene with energy contour plot of the bonding band and allowed k-lines for a metallic (5,5) and a semiconducting (10,0) nanotube 9
2.6 Electronic band structure and Density of states for a (4,2) nanotube 9
2.8 Density of states with bandgap renormalization 11
2.7 E_{22} (excitation) and E_{11} (emission) energies 11
2.9 Pillar spacing dependence of bridging probability of suspended SWCNTs grown on a square-lattice pillar array. 13
2.10 Schematics of the samples from Waseda University, Tokyo 14
2.11 SEM images of the samples (Trenches and pillars region) 14
2.12 SEM image of the sample from NRC 14

3.1 a) Schematics of the home built confocal microscope setup. b) Picture of the confocal microscope setup mounted on a dedicated optical table. 17
3.2 Schematic of the monochromator 18
3.3 Flowchart of the PLE automatization 23

4.1 Measured photoluminescence of a single SWCNT 26
4.2 Photoluminescence spectrum of a nanotube with a Lorentzian-fit 27
4.3 Setup modification in order to increase the beam size 28
4.4 Beam diameter measurement from the CCD image 29
4.5 Aging effect of nanotubes 30
4.6 Aging effect of nanotubes II. 30
4.7 Mapped samples with possible nanotubes 32
4.8 PLE map of a (11, 6) individual nanotube 33

B.1 GUI for the sample mapping automatization and stage control 78
B.2 GUI for the beam area calculation 79
B.3 GUI for the PLE mapping automatization 80

C.1 Table for assigning chiral indices to nanotubes based on their E_{11} and E_{22} energies 82

Abbreviations

BZ	**B**rillouin **Z**one
CCD	**C**harge-**C**oupled **D**evice
CNT	**C**arbon-**N**ano**t**ube
CVD	**C**hemical **V**apor **D**eposition
DLL	**D**ynamic **L**ink **L**ibrary
GUI	**G**raphical **U**ser **I**nterface
IR	**I**nfra **r**ed
InGaAs	**In**dium **Ga**llium **A**r**s**enide
MWCNT	**M**ulti-**w**alled **C**arbon **N**ano**t**ube
NA	**N**umerical **A**perture
NIR	**N**ear **I**nfra**r**ed
NRC	**N**ational **R**esearch **C**ouncil (Canada)
PL	**P**hoto**l**uminescence
PLE	**P**hoto**l**uminescence **E**xcitation
SEM	**S**canning **E**lectron **M**icroscope
SWCNT	**S**ingle **W**all **C**arbon **N**ano**t**ube
TB	**T**ight-**B**inding
USB	**U**niversal **S**erial **B**us
VHS	**V**an **H**ove **S**ingularity
ZF	**Z**one-**F**olding

Chapter 1

Introduction

The era of microelectronics is approaching its end, because scaling down electronic device dimensions is slowly reaching fundamental physical limits. In order to continue the envisioned path of Gordon E. Moore - who predicted that the computational power of semiconductor electronics will double, roughly every two years - a new technology has to be implemented. Nanotechnology provides a new perspective in fabrication, in comparison with the usual top-down scheme used in microelectronics, with the so-called bottom-up technique where nanostructures are assembled at nearly atomic levels. Working according to this philosophy, device dimensions can be scaled down to individual atoms.

The phenomena occurring at the nanometer scale cannot be described by classical physics anymore, but only by quantum mechanics. When the dimensions of objects shrink down to the nanometer scale, new properties emerge. One well-known example for this is given by graphite. Graphite (a stacking of two-dimensional, sp^2-bonded carbon layers) is known as a mechanically soft material, which is used in pencil leads. Now, if one imagines isolating a small sheet of a graphite monolayer (graphene) and roll it into a cylinder with a nanometric diameter, one will obtain a nano-object with amazing mechanical (tensile strength 80 times higher than high strength steel), thermal (better than copper) and electronic properties (completely described by tube geometry). Such objects called carbon nanotubes (CNT) have been discovered in 1991 by Sumio Iijima at NEC [1] and since then have led to an explosion of research activities in many labs worldwide. Furthermore, recent studies showed very promising optical properties which allow new applications and research in nanotube-based optoelectronics [2]. Their importance even grew in the recent past, since in the field of quantum information processing, carbon nanotubes are potential candidates that are able to naturally link solid state qubits used for information processing (such as single spins), with flying qubits used to transmit quantum information (photons) [3, 4].

The goal of my thesis is to get familiarized with carbon nanotube physics and related experimental techniques. Here the focus was put on the optical properties of single wall carbon nanotubes (SWCNT). I studied the photoluminescence (PL) of suspended carbon nanotubes on diverse samples. I implemented a system to facilitate the investigation of the photoluminescence excitation (PLE) mapping, which helped identifying the structure of the investigated nanotubes. Furthermore, I have come across an effect called bleaching, which corresponds to the decay of the PL signal. Based on the available literature, we addressed reasons behind this observation.

Chapter 2

Background

Carbon nanotubes discovered in 1991 by S. Iijiama [1], are important to scientists and engineers, because of their outstanding physical properties [5]. SWCNTs can be regarded as a sheet of graphene (one layer of graphite is called graphene) rolled into a cylinder. Among several nanostructures, carbon nanotubes have the smallest aspect ratio: in diameter they are only a few nanometers wide, but their length is usually a couple of μm, although it is possible to grow tubes that are as long as several cm [6]. Thus they are a quasi one-dimensional system, where quantum effects play a dominant role. The electronic structure of SWCNTs, that can be derived from the one of graphene, are unique in the sense that they are completely determined by the tube geometry, resulting in semiconducting or metallic character [5]. Furthermore SWCNTs are unique in the sense that they have prominent thermal [7], mechanical [7], chemical [8], electronic [5, 9] and optical properties [2]. In the next sections, the electronic and optical properties will be described in details.

2.1 Single wall carbon nanotubes

2.1.1 Atomic structure

The atomic number of carbon is 6 and as a group 14 element, four electrons are available to form covalent chemical bonds. The orbital structure of carbon consists of two electrons occupying the inner 1s shell. The other four can be arranged in 3 different ways, in other words, they can have three different hybridizations:

- sp^3 (4 covalent σ bonds), e.g.: diamond
- sp^2 (3 covalent σ bonds and 1 Π-bond), e.g.: graphene
- sp^1 (2 covalent σ bonds and 2 Π-bond)

In graphene and SWCNTs we have a sp^2 hybridization. In graphene, the honeycomb lattice builds up from three in-plane covalent σ bonds and a fourth π-bond, which is delocalised. Since SWCNTs are made up of graphene sheets rolled up into a cylinder, the three σ bonds are generally slightly out of plane due to the curvature and the π-bond gets delocalized along the tube walls. These σ bonds provide SWCNTs high mechanical strength along the tube axis - it has been measured that the Young's modulus for SWCNTs can be 80-times higher than that of high strength steel [10].

Now the nomenclature of nanotubes will be introduced. On Fig. 2.2, the honeycomb lattice structure of graphene is shown. Starting from one gridpoint (one C atom) using the linear combination of the basis vectors $\vec{a_1}$ and $\vec{a_2}$, we can build up one of the two sublattices of graphene. By repeating this process, starting from one of the neighboring atoms of the initial starting point, the complementary sublattice can be constructed. The chiral vector of a CNT is the circumferential vector along which the sheet of graphene is rolled up. We can express this chiral vector as a linear combination of the basis vectors, $\vec{a_1}$ and $\vec{a_2}$:

$$\vec{C} = \mathbf{n}\vec{a_1} + \mathbf{m}\vec{a_2}, \qquad \mathbf{n{,}m} \in \mathbb{N} \tag{2.1}$$

Many properties of SWCNTs can be derived from the chiral vector, therefore a comfortable notation is used to describe them: from the equation above, the two integers n and m, which are called chiral indices are expressed in brackets, such as *(n,m)*.

Depending on the chiral indices one can separate SWCNTs into three main groups (see Fig. 2.1):

- if m or $n = 0$, the nanotube is called **zigzag**
- if $n = m$, the nanotube is called **armchair**
- if $n \neq m$ the nanotube is called **chiral**

In real space the other important geometrical parameter of a CNT is the translational vector, which represents the smallest vector along the tube axis, which connects two equivalent lattice

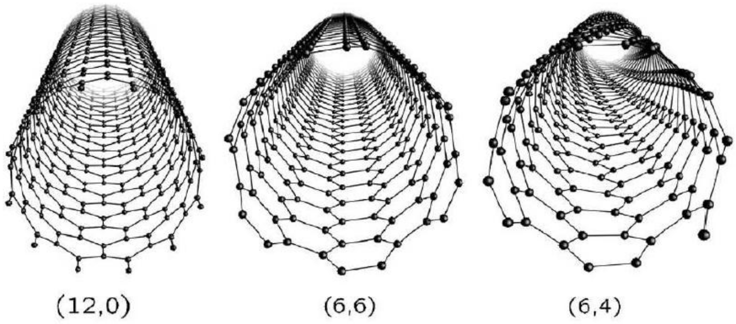

FIGURE 2.1: The three types of tubes: zigzag, armchair and chiral.

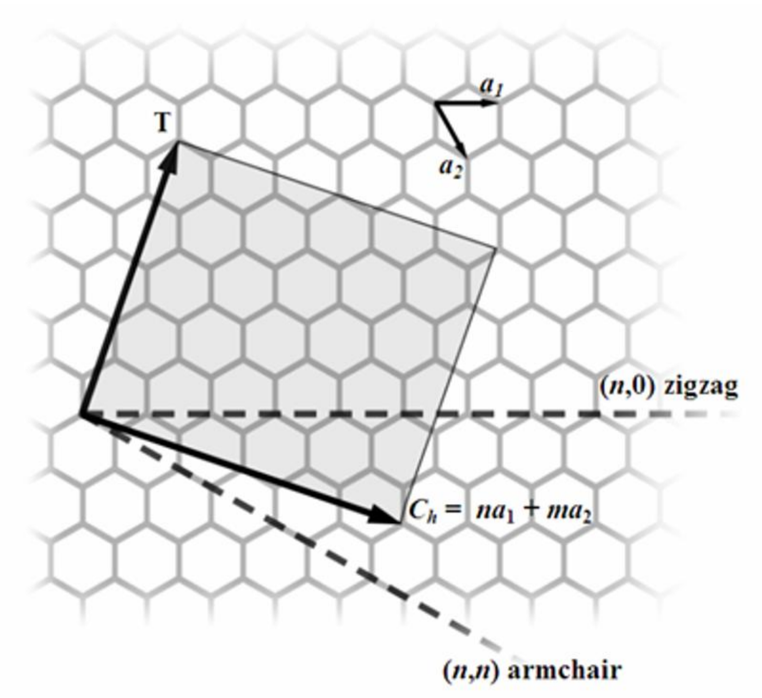

FIGURE 2.2: The chiral and translation vectors of CNTs

points. The translational vector is perpendicular to the chiral vector, and can be expressed again only with the chiral indices:

$$\mathbf{T} = t_1 a_1 + t_2 a_2, \qquad t_1 = \frac{2m+n}{N_R} \quad t_2 = -\frac{2m+n}{N_R} \tag{2.2}$$

where N_R is the greatest common divisor of $(2m+n)$ and $(2n+m)$.

The diameter of the tube can be expressed with only the chiral indices:

$$d_t = \frac{|C_h|}{\pi} = \frac{a}{\pi}\sqrt{n^2 + nm + m^2} \tag{2.3}$$

where a represents the lattice constant of the honeycomb lattice ($a = \sqrt{3}a_{cc} = |\vec{a_1}| = |\vec{a_2}|$, where a_{cc} is the the bond length with $a_{cc} \approx 1.42\mathring{A}$).

2.1.2 Electronic structure

As mentioned above, the electronic structure of a SWCNT can be derived from that of graphene. The unit cell of graphene is a rhombus that contains two carbon atoms as shown on Fig. 2.3 (a), in the dashed area. These two atoms belong to the two complementary sublattices that yield together the honeycomb lattice of graphene (A for sublattice A, B for sublattice B).

In the reciprocal space the corresponding BZ of graphene is similarly a honeycomb cell, as outlined red in Fig. 2.3 (b). The reciprocal lattice vectors are obtained from the condition $\vec{a_i} \cdot \vec{b_j} = 2\pi\delta_{ij}$:

$$\vec{b_1} = (\frac{2\pi}{\sqrt{3}a}, \frac{2\pi}{a}) \qquad and \qquad \vec{b_2} = (\frac{2\pi}{\sqrt{3}a}, -\frac{2\pi}{a}) \tag{2.4}$$

Using a tight-binding (TB) model, we can calculate the electronic structure of an infinite graphene sheet. The TB model is an approximation that only considers the interactions between neighboring atoms as perturbation and ignores the electron-electron interactions [11]. The calculations yield two wavefunctions, corresponding to the two complementary sublattices of electrons. From these wavefunctions the dispersion relation of graphene can be calculated [8] as later shown on Fig. 2.6 a.).

The electronic band structure is usually referenced in three high symmetry points, labeled on Fig. 2.3 with Γ, K and M (see Fig. 2.3 and 2.6 a.)).

According to [12], the unit cell of SWCNTs, spanned by the translational and chiral vectors, contains

$$N = \frac{2(m^2 + mn + n^2)}{d_R} \tag{2.5}$$

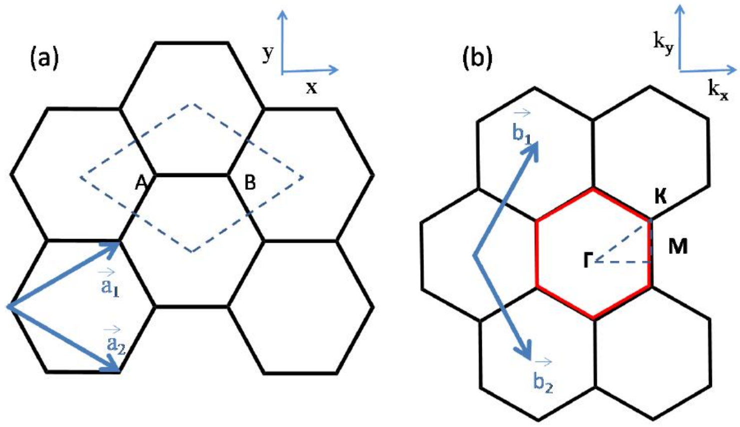

FIGURE 2.3: Unit vectors in real (a) and reciprocal space (b), [Adapted from "Unusual Properties and Structure of Carbon Nanotubes" [12]]

atoms which is much larger than that of graphene with only a two atom rhombus. This fact is important, because the number of electrons in the unit cell corresponds to the number of bands in the BZ, respectively.

In order to calculate the electronic structure of a SWCNT the zone-folding (ZF) approximation is used [11]. ZF approximation utilizes the fact that a rolled up sheet of graphene has periodic boundary conditions along the circumference of a SWCNT, thus the allowed wave vectors in direction perpendicular to the tube axis are quantized.

The periodic boundary condition is illustrated on Fig. 2.4. Due to the tubular structure of the SWCNT the wavefunctions at points P_1 and P_2 (with exactly one $\vec{C}$ between them) have to satisfy the following condition:

$$
\begin{aligned}
\Psi(\vec{x}) &= \Psi(\vec{x}+\vec{C}) \\
\Longrightarrow e^{ikx} &= e^{ikx+C} \\
\Rightarrow e^{ikC} &= 1 \\
\Rightarrow \vec{k}\cdot\vec{C} &= 2\pi n
\end{aligned}
\tag{2.6}
$$

Note: the second equation is a corollary of the Bloch-theorem. Plotting these allowed vectors for a given SWCNT onto the BZ of graphene generates a series of parallel and equidistant lines. The distance between lines can be calculated and is found to be $\Delta k = \frac{2}{d_t}$. The length, number and orientation of these lines depend on the chiral indices (n,m) [12]. These parallel cutting lines decide to which group a nanotube belongs to. For example the parallel lines shown in Fig. 2.5 are examples of allowed k modes for a metallic (5,5) armchair and a semiconducting (10,0) zigzag, respectively. Note: the white and black plots in Fig. 2.5 are equipotential lines of the electronic structure of graphene shown in Fig. 2.6 a).

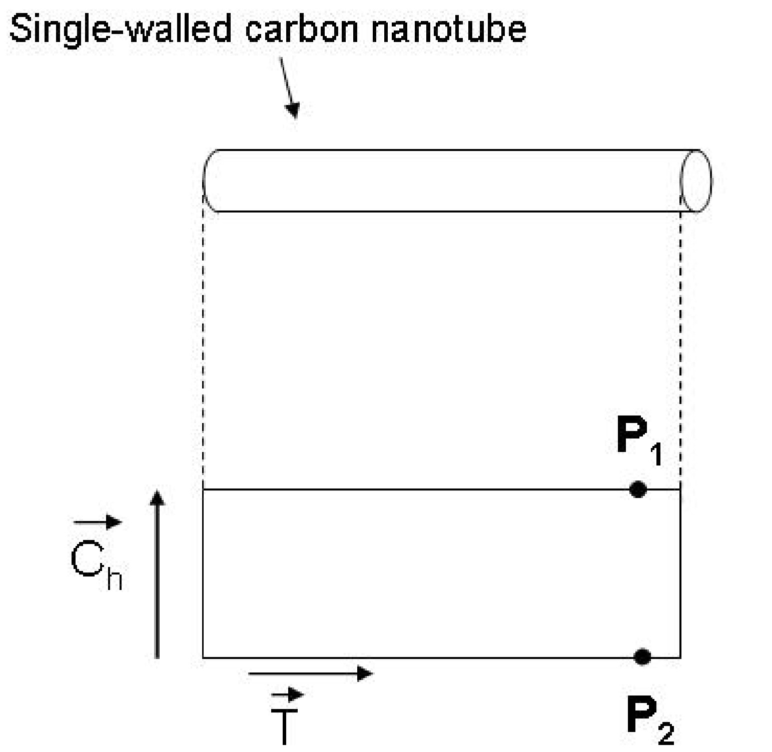

FIGURE 2.4: The periodic boundary condition using the ZF approximation

The basic idea behind the zone-folding approximation is that the electronic band structure of a specific nanotube is given by the superposition of the electronic energy bands along the corresponding allowed k lines as shown in Fig. 2.5, i.e. a pair of conduction and valence bands for each k line. Therefore the band structure in Fig. 2.6 b) correspond to the superposition of the line cuts in Fig. 2.6 a). If the K point is crossed by an allowed k line, the SWCNT is metallic, i.e the valence and the conduction band touch each other. It can be shown that a SWCNT is metallic if the condition $n - m = 3l$, with l an integer, is fulfilled.

When $n - m = 3 \cdot l \pm 1$, the allowed k vectors do not cross the K points, making the SWCNT semiconducting with a direct band gap. The band structure shown in Fig. 2.6 b) corresponds to the one of a (4,2) semiconducting SWCNT. It can be calculated that a semiconducting SWCNT has a band gap of:

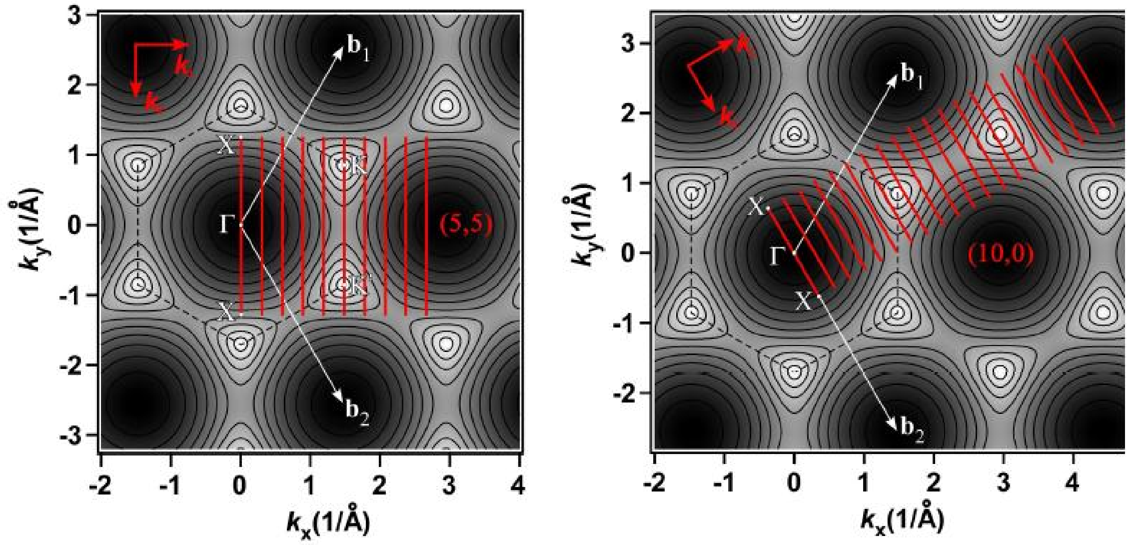

FIGURE 2.5: Reciprocal lattice of graphene with energy contour plot of the bonding band. The allowed k lines for SWCNTs arising from the quantization condition around the circumference: $C_h k = 2\pi q$ are drawn in red. $k_{||}$ and $k_{\perp}$ are the unit vectors in directions of C_h and T, respectively. Left: metallic (5,5); Right: semiconducting (10,0) nanotubes

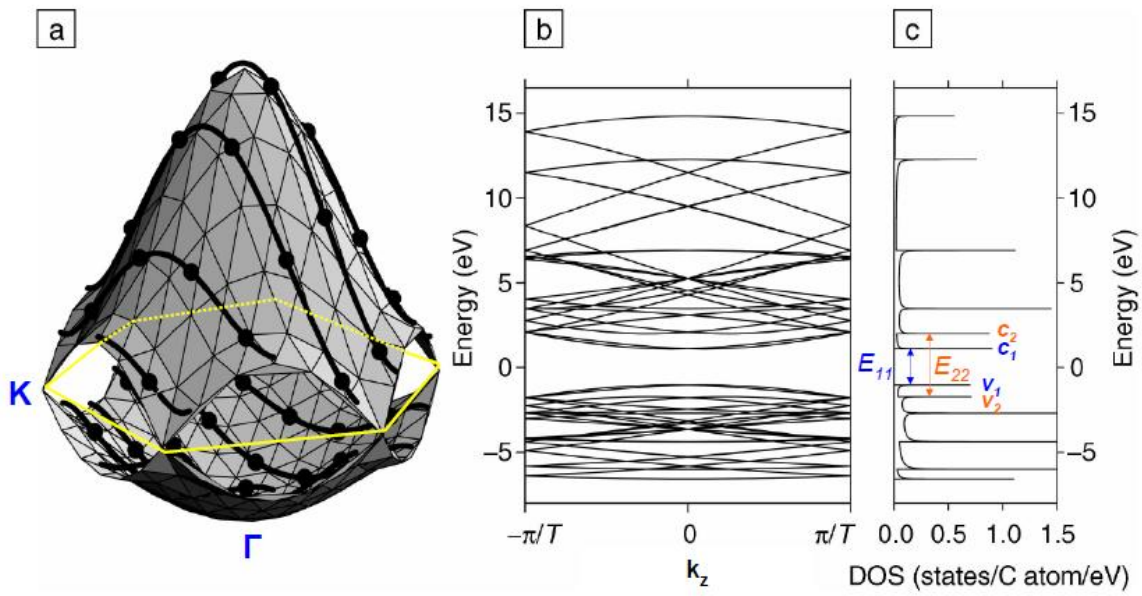

FIGURE 2.6: The dispersion relation (a), the electronic band structure (b) and the density of states (c) for a (4,2) carbon nanotube. The zero energy represents the Fermi level]

$$\Delta E_g = \frac{2a_{cc}}{d_t} \tag{2.7}$$

This $\frac{1}{d_t}$ dependence relies on the assumption of a linear dispersion cone around K-points in the BZ of graphene.

Due to its 1D character, the density of states (DOS) $\Delta N/\Delta E$ of SWCNTs is proportional to $(|\frac{\delta E(k)}{\delta k}|)^{-1}$ and diverges as $|E|/\sqrt{E^2 - (E_0)^2}$ close to band extrema E_0, as can be seen in the right hand panels of Fig. 2.6 c). These singularities in the DOS are called Van Hove singularities (VHS) and are important to understand the optical properties of SWCNTs.

It is important to understand that in the TB calculations above, the Coulomb interactions are omitted and only proceed with the calculations for the neighboring atoms in the lattice.

2.1.3 Optical properties

Photoluminescence (PL) is a process in which a substance absorbs one or many photons and then re-radiates photons. In quantum mechanics, this can be described as an excitation to a higher energy state and then a return to a lower energy state accompanied by the emission of a photon.

Photoluminescence in semiconducting nanotubes, which are direct bandgap systems, was observed for the first time in 2002 and opened the way to carbon nanotube optics [13]. In this experiment, carbon nanotubes were wrapped in a solution in order to separate individual tubes from bundles. After studying the PL signal of SWCNTs wrapped in various surfactants, it became clear that the wrapping material influences the PL spectra. Later it has been shown also that experimental conditions, such as temperature and pH of the solution alter the measured spectra of SWCNTs as well [14–16].

In this work, we used suspended nanotubes that were grown between structures that separate them from the substrate material (as described in the following section). Since suspended SWCNTs do not contact substrates or any surrounding medium and have been shown to emit intense and sharp PL peaks [17], they are ideal systems for the investigation of the optical properties of individual tubes. A quantitative comparison between the PL spectra from suspended nanotubes

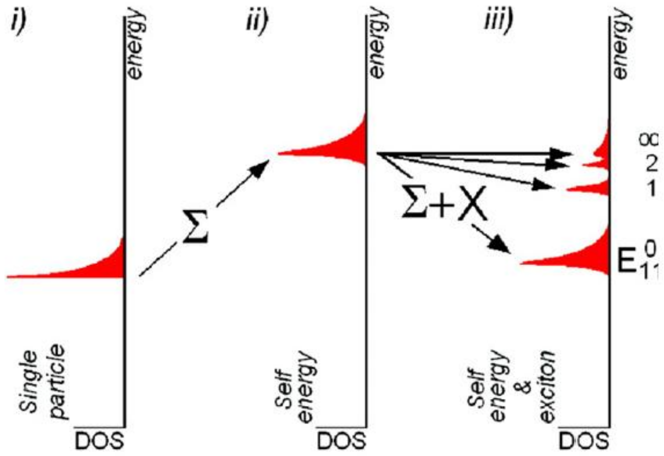

FIGURE 2.8: Density of states as suggested by i) TB approximation ii)TB approximation with Coulomb interaction iii) the normalized discrete excitonic states

and micelle-encapsulated nanotubes was reported by Jacques Lefebvre et al. and showed a red-shift of 28 meV in average for E_{11} and 16 meV for E_{22} for encapsulated nanotubes compared to air-suspended SWCNTs [18].

The absorption and emission energies are usually not the same in experiments. Nanotubes are excited with light polarized in the direction of the tube axis (antenna effect) in the E_{22} transition, and PL is measured from radiation in the E_{11} transition (Fig. 2.7). The E_{22}/E_{11} ratio from TB calculations is 2. However, experimental results yield a ratio around 1.8. This paradoxical observation is called the ratio-problem. This is due to the presence of excitonic states in the nanotube band gap.

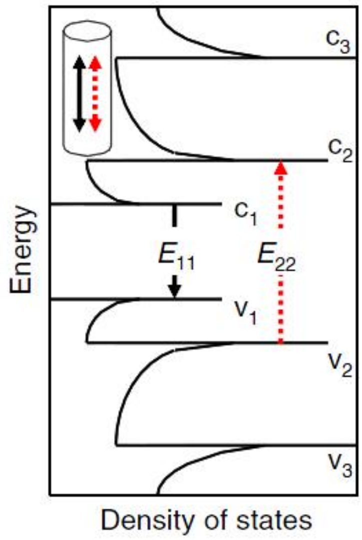

FIGURE 2.7: E_{22} (excitation) and E_{11} (emission) energies

An exciton consists of a photo-excited electron and hole bound to each other by a Coulomb interaction in a semiconducting material. In most bulk materials the binding energy is so low (magnitude of meV) that excitonic states can only be observed at very low temperatures. In the case of SWCNTs, because of their quasi one dimensionality, the confinement of particles is so high that the Coulomb interactions give rise to exciton binding energies of about 1/3 of the band gap [19], making the excitonic nature of SWCNTs observable at room temperature.

A more detailed view of the excitonic effects occurring in semiconducting nanotubes is depicted in Fig. 2.8. The emission energies in SWCNTs are the result of two nearly equally important Coulomb interactions, the self-energy which increases the band gap (ii) in the single particle model (TB) (i) and the excitonic binding energy which decreases it (iii) with a series of discrete excitonic states below the self-energy corrected continuum.

2.2 Synthesis

2.2.1 Sample preparation methods

Carbon nanotubes can be synthesized by various methods with arc-discharge, laser-ablation and chemical vapor deposition (CVD) being the principal ones. The SWCNTs used in this work have been produced by CVD method. In CVD, a flowing hydrocarbon gas is decomposed at a growth temperature between 500 and 1000^oC. The precipitation of carbon from the saturated phase in metal catalyst particles (generally Fe, Ni or Co) leads to the formation of a tubular carbon solid.

In the optical properties section it has been discussed that nanotubes lying on substrate material do not exhibit PL [17]. In the experiments I used suspended carbon nanotubes to avoid the environmental effect induced by the substrate material. The suspension of nanotubes can be done in multiple ways, but in my experiments I used mesa-structures where nanotubes are grown by CVD from catalytic particles lying on the mesa-structures. The mesa-structures used are trenches or pillars with different pitches.

The spacing between two mesa-structures (consecutive trenches, pillars) has influence on the tube bridging probability to a great extent. Fig. 2.9 shows the bridging probability (the percentage of pillar-pillar connections that have nanotubes growing between them) as a function of pillar-pillar distance. It can be seen that for increasing pillar-pillar distance, the number of connections decreases exponentially. Note: if the pillar-pillar distance approaches zero, the probability of bundled nanotubes rises dramatically. Bundles of nanotubes generally quench the PL and are therefore to avoid. It is preferable to have a larger distance between the pillars, thus a lower bridging probability, but less entanglement. For a more in-depth analysis of the synthesis of the synthesis of suspended tube samples, please refer to [20].

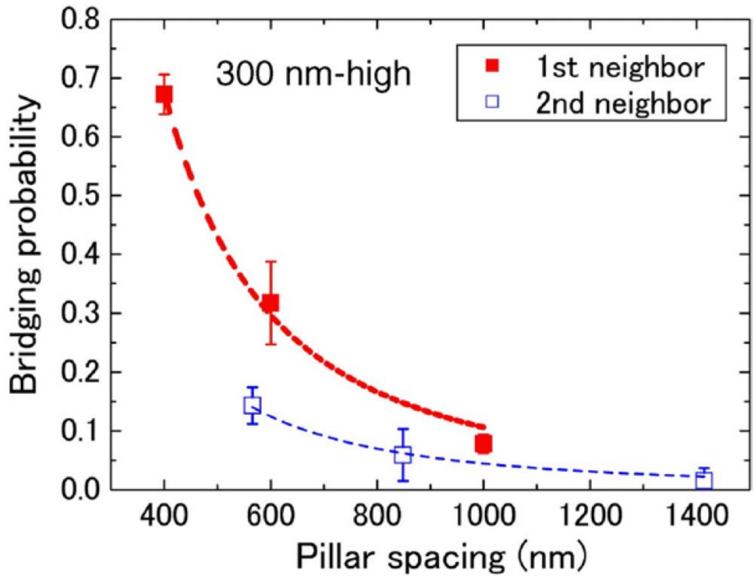

Figure 2.9: Pillar spacing dependence of bridging probability of suspended SWCNTs grown on a square-lattice pillar array.

While the growth techniques described yield a high amount of high-quality nanotubes, the formation mechanism of SWCNTs is still unknown. Previously the growth direction and speed was credited to the flow of gas on the sample, but a series of experiments done by at Tokyo University showed that SWCNT growth can occur even perpendicularly with respect to direction of the gas flow [21].

2.2.2 Samples used in the experiments

In our experiments the samples were provided from three research groups: from Waseda University, Tokyo; from NRC, Canada and from TU Delft. The fabrication method applied was CVD in all three cases. The pattern profile differed from sample to sample.

The samples provided by Prof. Y. Homma (Waseda University, Tokyo) have a Si substrate as base material and the SWCNT are grown on the surface with a CVD technique. The group used Co catalysts on their patterns to induce nanotube growth. Different patterns were defined with chemical etching . As Fig. 2.10 shows, the samples have 10 regions consisting of trenches (Fig. 2.11 a)), while the other consists of pillars (Fig. 2.11 b)).

It is important to note that one of the three samples provided by Waseda University has been investigated by SEM imaging (as seen on Fig. 2.11). It is known that high-energy electron irradiation on a nanotube causes structural damage due to a ballistic ejection of carbon atoms.

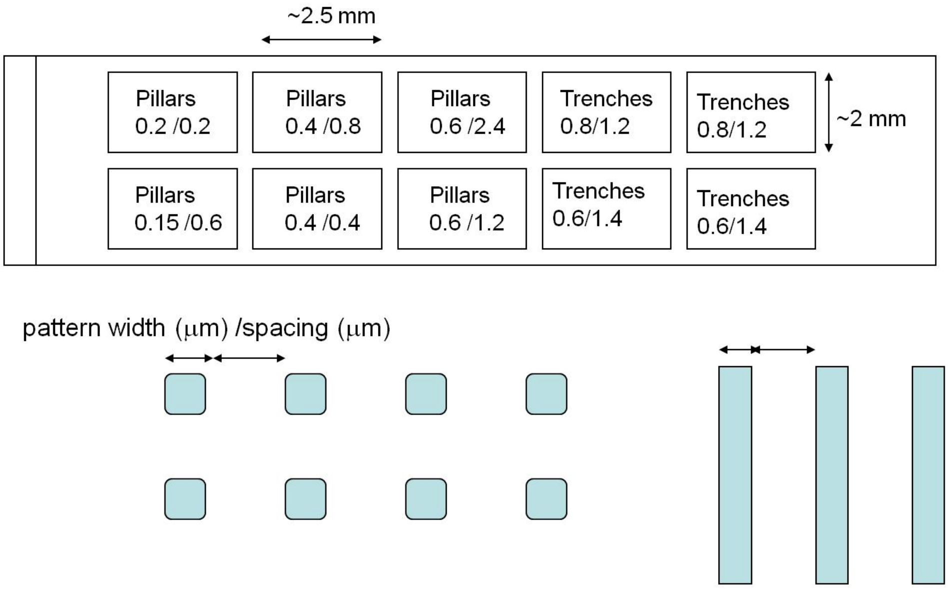

FIGURE 2.10: Schematics of the samples

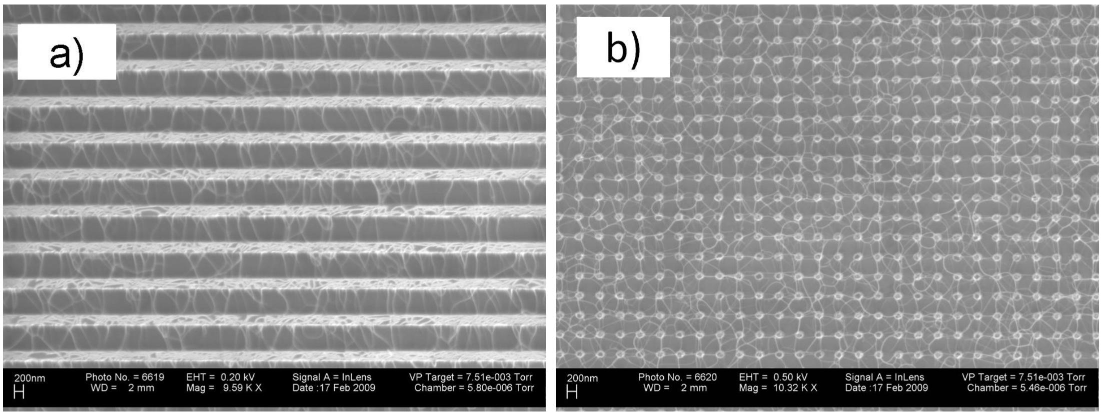

FIGURE 2.11: SEM images of the samples

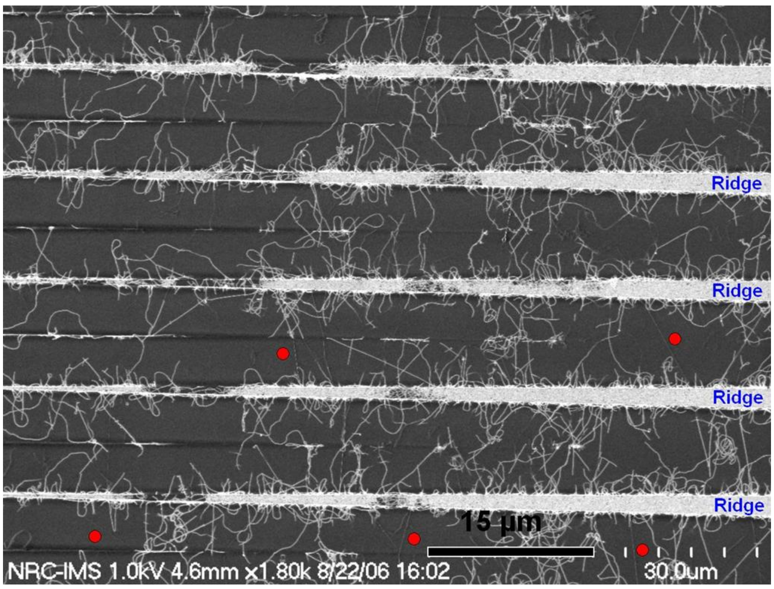

FIGURE 2.12: SEM image of the sample

Experimentally it has been shown that this effect leads to the decrease of the PL of nanotubes. However this phenomenon is paradoxical: since the threshold of ejection of carbon nanotubes is 86 keV, whereas SEM imaging bombards the sample only with 0.5-25keV electrons [22]. Interestingly it has been reported that these structural defect heal at room temperature, depending on the diameter of the nanotubes [23].

The samples from NRC are similar to the samples from Waseda University. They are made of trenches only, as shown on the SEM image on Fig. 2.12.

The last group of samples were designed and synthesized by the QT group of TU Delft. The patterns on the sample consists only of trenches, which are spaced periodically.

Chapter 3

Experimental design

3.1 Equipment

3.1.1 Confocal microscope setup

Suspended individual carbon nanotubes emit light in the NIR when excited in their E_{22} transitions. In order to study this effect we built a dedicated confocal microscope setup for near infrared optics, optimized for PL detection of nanotubes. The schematics as well as a picture (early stage) of the setup are shown in Figure 3.1 a) and b), respectively.

The excitation light from different sources (described in section 3.1.3) is guided to the setup through a single or multimode fiber (1). The beam is then collimated by an aspheric lens and directed towards a 10:90 neutral density filter (2) such that 10% of the beam intensity is focused on the sample (mounted on a piezo x-y-z stage from Newport described in section 3.1.4 (5)) through an IR objective with a NA of 0.42 and a working distance of 20 mm (4). The 90% of light reflected by the density filter reaches a powermeter (described in section 3.1.5) in order to control the power on the sample. The PL collected from the sample, as well as a reflected fraction of the laser light goes back through the microscope objective and about 90% of the intensity is reflected by the neutral density filter (2) in the spectrometer (7) direction, where the sample reflected laser light is blocked by a long wavelength pass filter with a cutoff wavelength of 1150 nm (6). The sample is imaged using white light (11) directed towards the sample through a flip 45:55 pellicle beam splitter (3) and through the objective (4). The light reflected from the sample is reflected by a fixed pellicle beamsplitter (8) and focused (9) on a

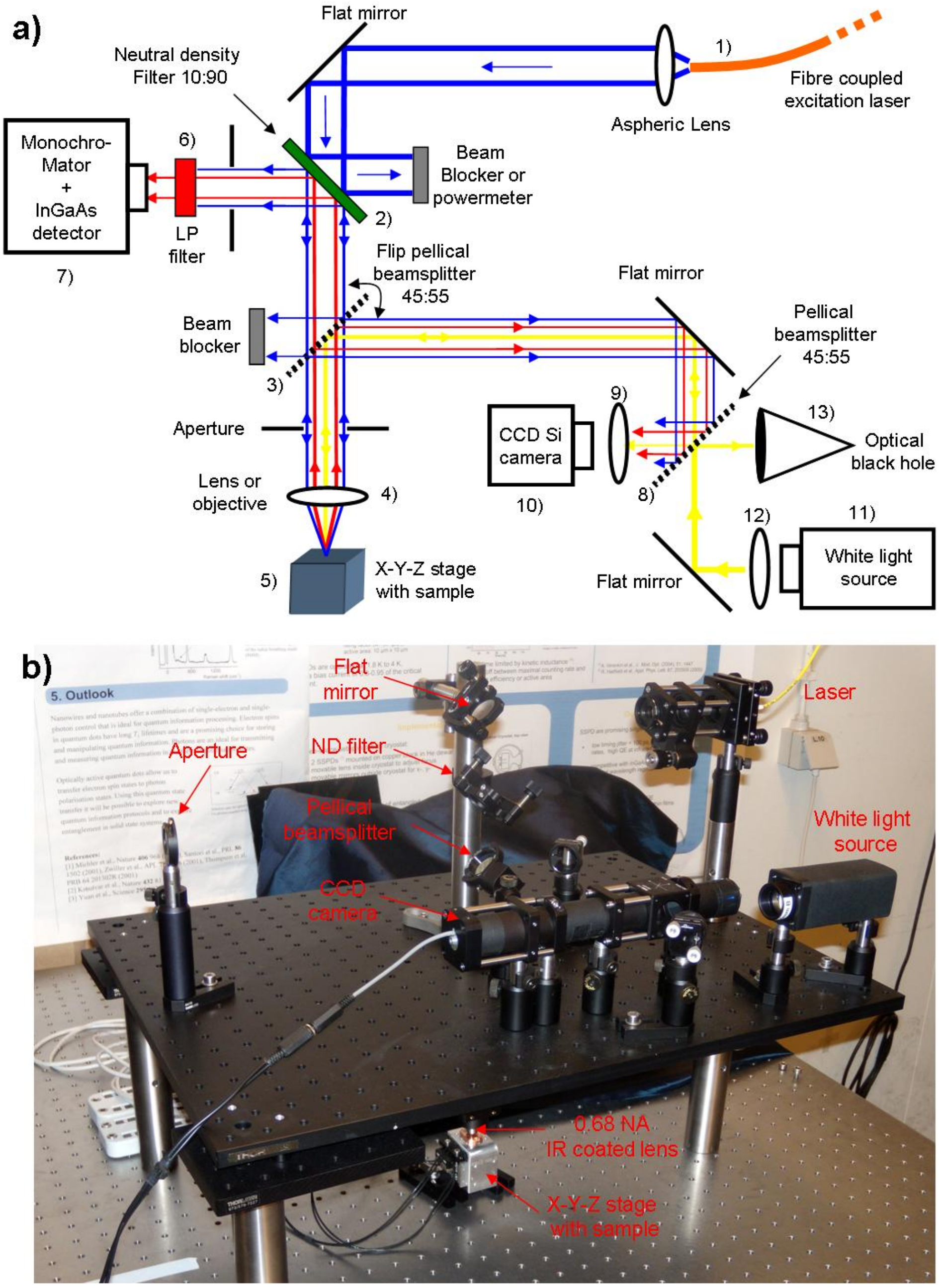

Figure 3.1: a) Schematics of the home built confocal microscope setup. b) Picture of the confocal microscope setup mounted on a dedicated optical table. The monochromator with a fitted InGaAs detector lies on the same table, optically aligned with the setup (on the left, not shown on this picture)

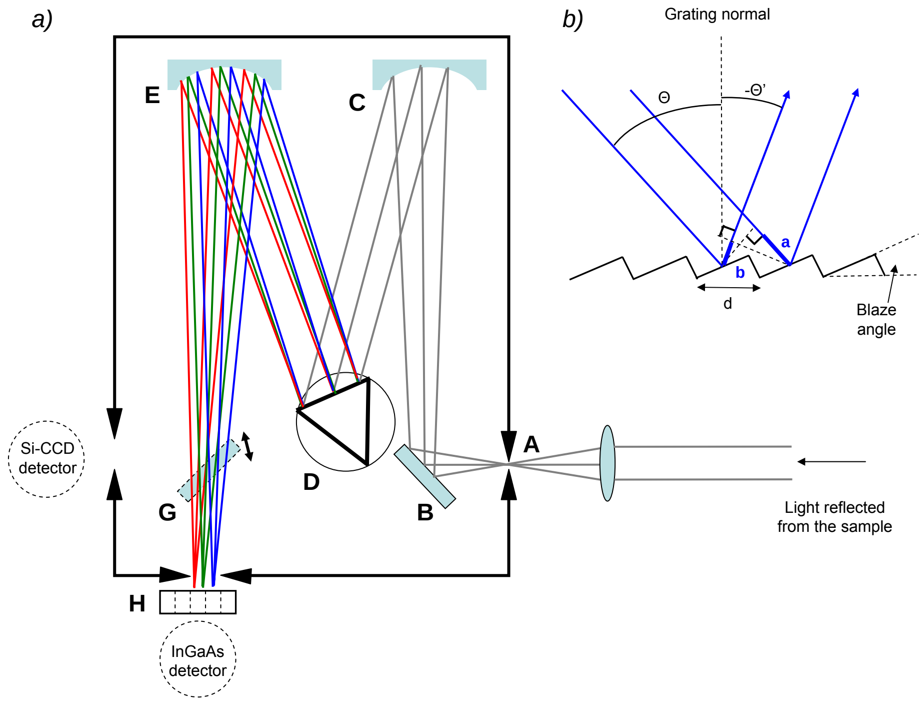

FIGURE 3.2: Schematic of the monochromator

CCD camera (10). The quality of the image is dramatically improved by the use of an optical ”black hole” (13) to avoid that light coming directly from the white light source disturbs the image. The spectrometer is connected to a PC, via USB connection, in order to control the mirrors and gratings inside.

On Figure 3.1 b) the setup in its earlier state can be seen. The sample scanning is made possible through a moving stage that has three degrees of freedom, and which is controlled via the serial COM port of a PC. In general most of the control software was written in MATLAB.

3.1.2 Spectrometer

The detection part of our setup consists of a nitrogen cooled InGaAs detector array fitted to a monochromator. The Princeton Instruments Oma V detector [24] consists of a linear array of 512 InGaAs photodiodes able to detect light with a wavelength ranging from 0.8 μm to 1.7 μm. The monochromator is a Spectra Pro 750 with an aperture ratio $f/9.7$ corresponding to a NA of about 0.05, or an acceptance angle of $2 \cdot 2.9^o$[1]. A schematics of the spectrometer and the InGaAs detector can be seen in Fig. 3.2 a.).

[1] $\frac{f}{\#} = \frac{1}{2 \cdot NA}, NA = n \cdot \sin\theta$

The fundamental operation principle of the spectrograph is described in the following paragraph. A collimated light beam coming from the confocal microscope setup is focused at the entrance slits (A) by an $f = 5$ cm lens. The choice of the focal length of the lens is dictated by the f-number of the monochromator (f/9.7) and the diameter of the collimated light beam. The light rays that passed through the slit are directed through a planar mirror (B) onto a parabolic mirror (C), which reflects the incident rays in a collimated beam. This collimated beam is then diffracted by a grating (D), which disperses light with different wavelengths with different corresponding angles.

The dispersed light rays with different wavelengths are finally focused by a second parabolic mirror (E) on different spatial locations along the InGaAs array (H). Thus a spectrum of the light intensity as a function of the wavelength can be obtained. Note: in our setup two detectors were available: an InGaAs detector and a Si CCD detector. The selection of the detector was made through a flip-mirror (G).

A more detailed description of gratings physics is given in this paragraph. The cross-section of a ruled grating surface with incident and diffracted rays is shown in Fig. 3.2 b.). In principle a ruled grating consists of a substrate material (typically crystal silicon substrate or fiber-glass) with a large number of parallel grooves, coated with a reflecting material such as aluminium. On the figure two rays of the same wavelength travel parallel to each other and impact the grating on two neighboring grooves. A maximum of intensity occurs, when the two refracted rays have the same phase. Therefore, the location of intensity maxima can be calculated, from the path difference between the rays:

$$\begin{aligned} n \cdot \lambda &= a - b = \\ &= d \cdot sin(\theta) - d \cdot sin(\theta') \\ &= d(sin(\theta) - sin(-\theta')) = \\ &= d(sin(\theta) + sin(\theta')) \end{aligned} \tag{3.1}$$

Where d is the groove spacing, λ the wavelength, θ and θ' the angles measured from the grating normal for the incident and reflected angles, respectively and n is an integer, which corresponds to different diffraction orders of a grating (we usually only work with the first order, $n = 1$).

Eq. 3.1 is known as the grating equation. Dispersion, which refers to the angle in which the same spectrum is spread can be deduced quantitatively from the grating equation and is found to be inversely proportional to d, the grating constant.

$$\frac{d\theta'}{d\lambda} = \frac{m}{d \cdot \cos\theta'} \tag{3.2}$$

Thus, for a grating with a high (low) groove density the dispersion will be large (small), and consequently the resolution will be high (low) but with lower (higher) intensity. The blaze angle (Fig. 3.2 *b.)*) determines the efficiency curve of a grating. For example, a grating "blazed" at 1,3 μm will have its maximum efficiency at this wavelength.

In our setup, 3 different gratings are mounted on a turret (D). These gratings have the following parameters: 1) 85 g/mm, "blazed" at 1,3 μm, 2) 600 g/mm, "blazed" at 1,6 μm and 3) 1800 g/mm, "blazed" in the visible, with the software (WinSpec), a setting is available to change the grating in use and the center wavelength of the detected spectrum.

The InGaAs detector, consisting of an array of 512 photodiodes, is characterized by different parameters. An important one is called dark noise. There are two sources of dark noise: first, the reverse bias leakage current, which is due to minority carrier diffusion and secondly thermally activated carriers.

To decrease the dark noise, the array is cooled down via a cryostat filled with liquid nitrogen. Once the sensor's temperature drops to approximately -100 ^0C the system is ready for measurement.

Statistically dark noise is random and since its distribution is even, it can be subtracted from the measured signal in order to increase signal to noise ratio. For this a background of the dark noise is recorded before measurement (slit closed) and the software automatically subtracts it from the signal.

An important measurement parameter is the integration time, which determines the amount of time elapsed to record a spectrum. Since dark noise is completely random an integration time twice as long increases the signal to noise ration by a factor of $\sqrt{2}$, thus long integration times (1 minute or more) are helpful, to get "cleaner" SWCNT PL spectra. Shorter integration time is used when scanning a sample, in order to shorten the measurement time.

3.1.3 Lasers

In the laboratory I had access to two laser sources: a Mira 900 and a Spectra Physics 3900. The Mira 900 is a mode locked ultrafast laser that uses Titanium:sapphire as the gain medium. It is tunable from 710 to 1000 nm. There are two modes of operation for this laser: the low power configuration uses 8 Watts and the high power 12 Watts from the pump laser (Verdi). The emitted laser power is 500 mW and 1100 mW, respectively. The output polarization is horizontal in every mode. The 3900 manufactured by Spectra Physics also uses Titanium-doped sapphire as the gain medium. The peak intensity is reached at 790 nm. An average output power of 2.5 W can be reached with a pump power of 20 W. The polarization is horizontal for all operation modes.

It can be shown that the 700 nm-1000 nm range of these lasers fits with the E_{22}, transitions of small diameter (< 1.2 nm) SWCNTs emitting in the available InGaAs detection range (900-1600 nm).

3.1.4 Motors

During my experiments I used two types of motors: 3 piezo-motors (from Newport) and a stepper motor (from Standa). The piezo-motors have been used to manipulate the sample stage with a 50 nanometer precision, although this was not a reproducible movement, since the motor operated in an open-loop. The stepper motor was used to change the wavelength of the laser. Both of the motors were automated from a Matlab routine via the serial interface of the computer. The implementation codes can be found in Appendix A.

3.1.5 Power meter

A power meter (Thorlabs PM100) was used in order to measure the laser power density on the samples. The readout range of the power meter depends on the sensor in use. In the laboratory we had access to sensors to read power ranging from 50 nW up to 20 W, with a 12 bit resolution. The read-out of the power meter as well as the tuning of measurement parameters (e.g λ range) could be computer-controlled via the serial port.

3.2 Description of the measurement procedures

The focus of my thesis is the optical investigation of carbon nanotubes. My tasks included:

- Find a nanotube manually, for the purpose of optimizing the detection parameters.
- Sample mapping, in order to get a map of the position of optically active SWCNTs on a large area (mm $\times$ mm).
- PLE mapping, in order to assign chirality of a SWCNT

These three tasks are implemented in two Matlab GUIs (one for manual search and sample mapping, the second one for PLE mapping) that I want to describe below briefly.

3.3 PLE mapping automatization

As described earlier in the theory section, semiconducting nanotubes can be excited in their E_{22} transitions and emit light in the E_{11} transition. When the excitation photons have exactly the same energy as the E_{22} transition, we say that they are resonant. In this case the emission intensity is the highest. It would then be very useful to record the emission wavelength and intensity as a function of the excitation wavelength. Such a measurement output is called a photoluminescence excitation (PLE) map.

To realize this, I controlled the excitation wavelength of the Mira 900 or the Spectrum Physics 3900 lasers with a step motor (section 3.1.4) connected to the micrometer control of the laser with a belt. The step-motor is controlled via the USB port by a control box. The range and velocity wavelength change rate are controlled from a Matlab GUI that I wrote, using a DLL package to command the control box. I calibrated the motor so that the wavelength change is a linear function of the number of steps. Since the output power of the laser is not constant over the full NIR range, I had to normalize each recorded spectrum. For this I read the power on the sample (indirectly from the reflected power, see Fig. 3.1 a)) from the power meter via the RS-232 port. A flowchart of the GUI program is shown in Fig. 3.3.

Upon initializing, the user has to reset the current stepper motor position, which will be associated with the current wavelength. From this point on the program is able to calculate the current wavelength, based on the ratio that I measured (the slope of the step position vs. wavelength

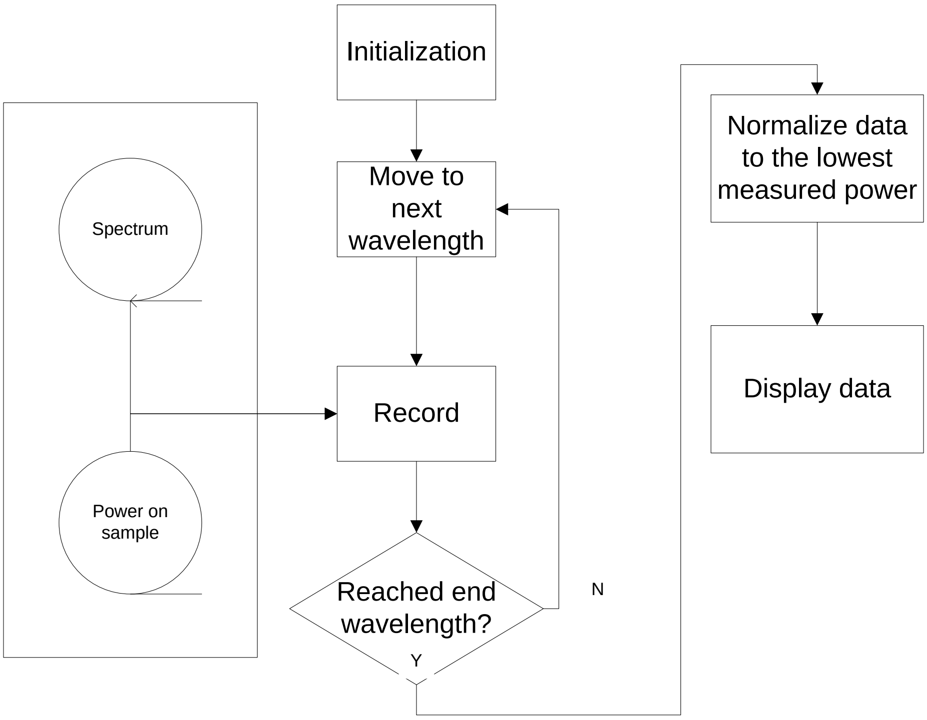

FIGURE 3.3: Flowchart of the PLE automatization

line). The program allows the user to move to a certain wavelength, or to a certain position. The PLE map generation needs only 3 parameters: the starting and the end wavelength and the step size, which determines the resolution of the PLE map.

To calculate the power density the area of the beam had to be calculated. For this I wrote a software that utilizes the CCD camera image through an image processing algorithm. The image taken by the CCD camera is first converted, so it can be manipulated in MATLAB. In Matlab the user has to select an X or Y coordinate, along which a cross-section will be made through the beam. From the cross-section the beam's diameter can be determined in the scale of pixels. To convert this to a micrometer scale I used a calibration sample, in order to find out the conversion ratio.

3.3.1 Manual search and sample mapping

As described earlier, the sample with suspended nanotubes is positioned on a x-y-z stage actuated by three piezo steppers from Newport (AG LS-25) with a specified minimum step size of 50 nm. These steppers are operated in open loop and the step size is therefore not reproducible.

The GUI that we developed in Matlab to control the piezo steppers via the serial communication port for manual search and sample mapping is shown in Appendix B. Each direction can be set with three different step sizes (low, medium, high). The motors are equipped with an odometric position feedback, therefore we are able to track the movement of the stage precisely (after compensating the motors). After locating a starting point for our scan we have to reset the positioning so we know how far we are on our sample. This has to be done, because it is not possible to record a spectrum and to image with the CCD camera at the same time.

For sample mapping, the user can input the amount of steps to be scanned in X and Y directions, with the amount of "piezo-steps" in each step. Since the backward and forward direction steps are intrinsically different, we have to compensate to minimize the drift in imaging. The scanning takes place first from the left to right than from the right to the left, in a meander shape. The sequence consists of 4 stages: first the motors move to the next location, then the program waits for the motors to reach their destination and for all the drift effects to die out. Next, the spectrum gets recorded, therefore the program has to maintain communication with the spectrograph's motherboard. This is done via an ActiveX Server initialized from Matlab. Depending on the integration time set in WinSpec (the GUI for the control of the spectrograph, provided by Princeton Instruments), the program waits for the execution and after waiting and additional time, the sequence repeats itself.

When the final destination is reached the program stops. All the spectra taken during the process are stored in a folder, in a format that is custom to Princeton Instruments, with the extension .SPE, which contains a header with information about the whole experiment (date, sensors used, grating and mirror positions, dark noise, etc.).

The final data set of a sample mapping consists of a 3-dimensional matrix with the position in x-y and the spectra in z. It is then possible to scroll through in z to obtain x-y maps at different emitted wavelengths.

Chapter 4

Results

4.1 Locating SWCNTs

My first task was to locate SWCNTs on different samples. For this, I used the equipment described in Chapter 3. Although the density of nanotubes on the sample is relatively large, only a very small proportion can be detected. This is due to the following reasons:

- In average, only 2/3 of nanotubes in a batch are semiconducting, i.e. having a band gap. The other 1/3 are metallic and can only emit light in very special conditions [25].
- Among the 2/3 of semiconducting nanotubes, only small diameter ones (< 1.2 nm approximately) emit in the detection range (900 nm - 1600 nm) of our InGaAs array. We do not have a statistical distribution of the tube diameter available, but based on the experience in growing tubes in the group, it is thought that the probability to get $d < 1.2$ nm is less than 10%.
- Furthermore, as a function of the trench width or pillar pitch, the probability to get entangled tubes or bundles is relatively high [22]. If small diameter semiconducting tubes are bundled with larger diameter tubes, it has been shown that the excitons generated in these tubes diffuse to larger diameter ones and recombine by emitting photons out of detection range. If metallic tubes are present in the bundle, they can quench the PL [26].

All these points together make the detection of a single SWCNT very challenging.

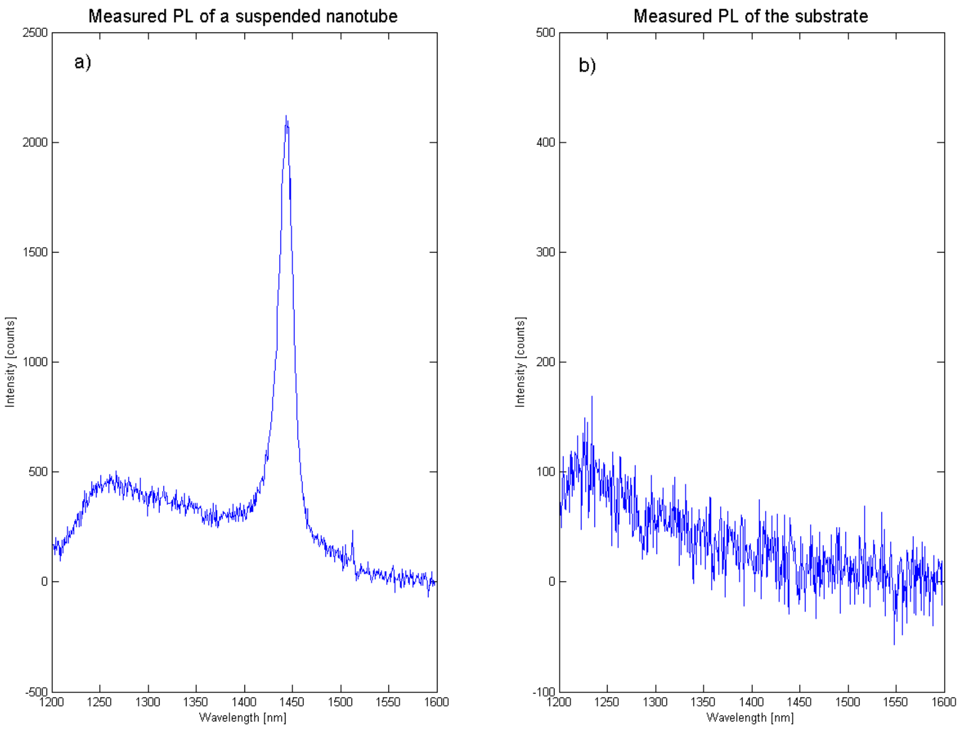

FIGURE 4.1: Measured photoluminescense. a.) SWCNT found b.) no tube in the spot area

The search for a SWCNT was done manually in a first step. Figure 4.1 shows two spectra: a) An optically active nanotube was lying under the beam, (integration time 60 seconds). Next to the PL signal of the nanotube around 1440 nm, the edge of a peak can be seen around 1250 nm, which is due to PL from the Si substrate luminescence. SWCNTs luminescing below 1250 nm are therefore more difficult to detect. On b), a spectrum, where no optically active tube is lying in the beam. Only the PL from the substrate is visible.

Figure 4.2 a) shows a PL spectrum from an individual SWCNT with a higher integration time in order to increase the signal to noise ratio (60 s integration time). The spectrum shows a peak at 1455 nm corresponding to 852 meV (conversion formula in footnote)[1]. This peak shows the characteristic asymmetric line shape with a sharp rise on the low energy side (larger wavelength) and a more gradual fall off to zero at high energies (lower wavelength), consistent with the shape of van Hove singularities (VHS) in the joint density of states [27]. From a lorentzian fit, the full width at half maximum is found to be about 16 meV. The resonant nature of the photoluminescence intensity can be checked by varying the Ti:sapph laser wavelength while

[1] $E[eV] = h\nu = \frac{hc}{\lambda} = \frac{4.136 \cdot 10^{-15} eV \cdot s \cdot 2.998 \cdot 10^{8} m \cdot a^{-1}}{\lambda(m)} = \frac{1{,}23}{\lambda}$

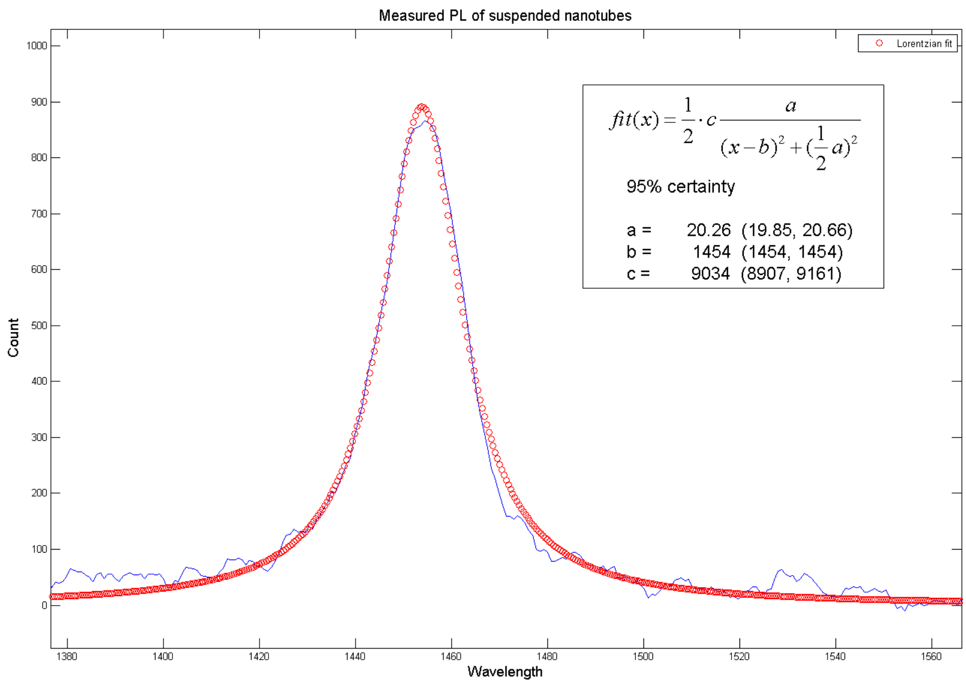

FIGURE 4.2: Photoluminescence spectrum of a nanotube with a Lorentzian-fit

recording the emission spectra. The result of these measurements is called a PLE map. PLE maps are very useful to assign the chirality of nanotubes and will be discussed later in this chapter.

Once a nanotube is found, the focus distance, i.e. the distance between the tube and the objective has to be optimized in order to get the maximum signal in the monochromator. This corresponds to a nearly collimated beam before the monochromator lens (see Fig. 3.1 a)). The focus was first set to get the sharpest white light image. From basic optical knowledge it is known that light with longer wavelength focuses further from the lens. For this reason I increased the distance between the stage and the objective until I reached the maximum intensity. In terms of piezo-steps, this distance has been found around 30 and 55 in low resolution.

I repeated the optimization process on more than 10 nanotubes distributed over the sample. Initially the beam size was too small, making the search for a tube even more difficult. One of the improvements that had to be made to the setup was the implementation of a lens positioned at twice the focal length from the objective aperture. This leads to a defocusing of the beam, giving rise to a larger spot size. Figure 4.3 shows the setup alteration that has been implemented in order to increase the probability of presence of a nanotube under the laser spot during scanning.

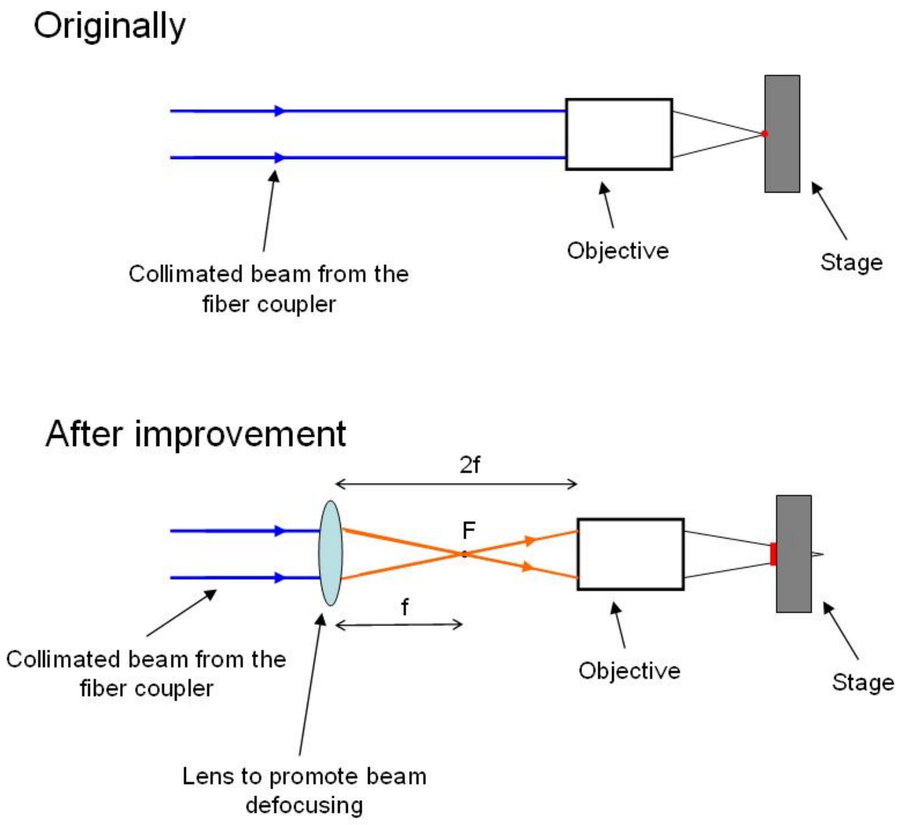

FIGURE 4.3: Setup modification in order to increase the beam size

An important parameter for PL measurements is the power density applied on the sample. To precisely tune the power density, the power on the sample and the size of the laser spot have to be measured.

The power can be measured precisely using the parameters described in Chapter 3. The problem arises, when the beam area has to be measured. The measurement of the laser spot diameter is made from a line cut in the CCD camera image processed in MATLAB, as show in Fig. 4.4. The laser power had to be decreased to minimum to ensure that the CCD camera did not saturate. The MATLAB algorithm that I wrote can be found in Appendix A (getbeam.m). In the case where the beam was defocused, we measured a beam diameter of 11.9 μm, while the original beam size was measured to be only 8 μm for a wavelength of 820 nm with the laser coupled to a multimode fiber. Thus the optimal power on the sample in these conditions was calculated to be 0.63mW and 1.41mW, respectively.

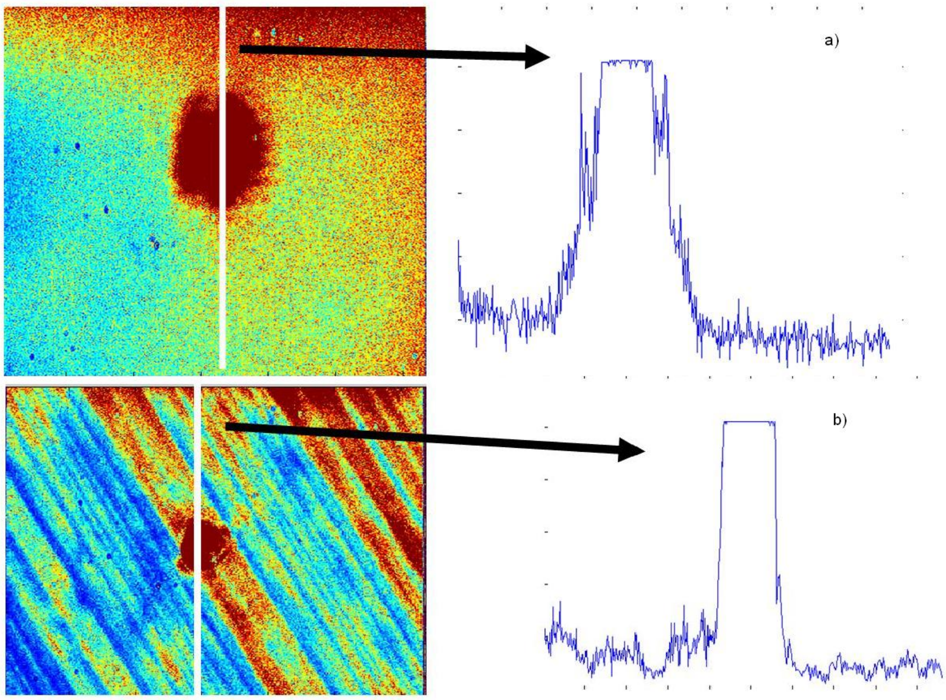

FIGURE 4.4: Beam diameter measurement from the CCD image a) beam with lens, b) beam without lens

4.2 The aging effect of SWCNTs

During the measurement process, I came across a noticeable effect: the decay of the PL intensity over time. This effect has been observed for all investigated nanotubes on different samples. Fig. 4.5 shows a graph of the intensity in counts/s as a function of the time for a SWCNT excited at 860 nm (close to resonance) under a power density of about 0.44 kW/cm^2. The intensity decreased by a factor 10 after about 3h of excitation.

The plot in Fig. 4.6 shows the time elapsed under continuous excitation at 860 nm to reach an intensity decrease of a factor 10 as a function of the power density for four nanotubes. It suggests a quicker decay with higher power density. A similar behavior has been observed by another group at Queen University in Canada. They observed differences in PL decay as a function of the relative humidity. A low relative humidity of 15-20% reduced the incidence of aging [28].

Another reported source of PL decay is the collapse of the nanotubes under purely optical forces (with power densities of the order of MW/cm^2), either from photon momentum transfer,

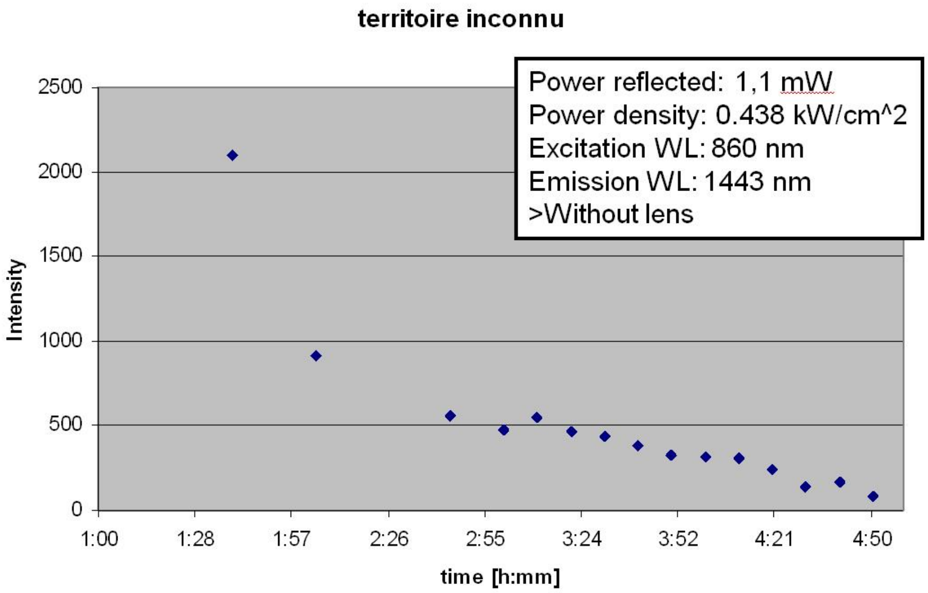

FIGURE 4.5: The degradation of the PL signal from an individual SWCNT over time

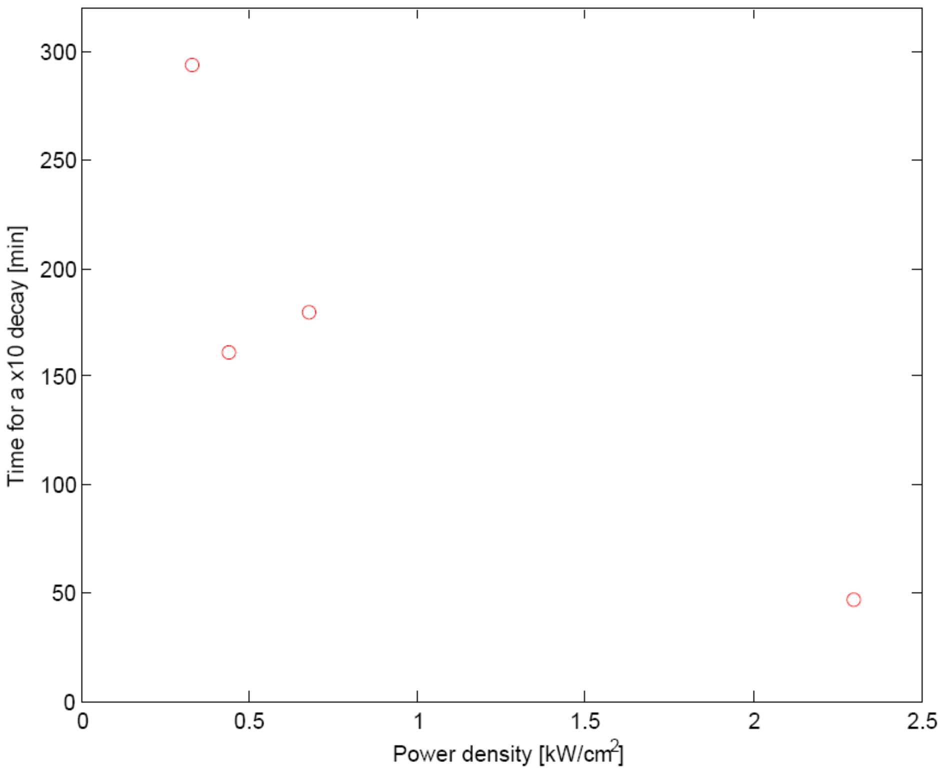

FIGURE 4.6: The time elapsed under continuous excitation at 860 nm to reach an intensity decrease of a factor 10 as a function of the power density for four nanotubes

or from the induced dipole gradient [29]. From the reported power densities, we think that in our case the effect of humidity is more likely to be the main source of decay.

This effect prevents long measurements like PLE maps (see next sections) and has to be minimized in future experiments. This can be done by working with the sample in a vacuum chamber purged and filled with a nitrogen atmosphere for example [28].

4.3 Sample mapping

Here the goal is to provide a map of the position of optically active nanotubes on a macroscopic area (typically a few mm^2) using the program described in section 3.2.3. A good trade off between the time needed to record such a map and the spatial resolution has to be found. In the map shown in Fig. 4.7, the scanning area is about 200 μm $\times$ 200 μm, with approximately 3500 points in each x and y directions. The integration time is set to 2 s (which is the inferior limit in order to clearly identify the PL of a tube out of the noise) and the excitation wavelength is set to 820 nm. The choice of the excitation wavelength determines the tube chirality we are looking for. 820 nm does not correspond to any resonant wavelength, but doing so we include several chiralities with a resonant excitation close to 820 nm, i.e. (13,2), (12,4) and (11,6).

Since the integration time is set at the limit of detection in the map shown in Fig. 4.7, it is difficult to attribute the spots to SWCNTs. Also, at this stage of development the setup could not be used yet to trace back the tubes since the piezo motors are working in open loop, i.e. the step size is not reproducible. Another difficulty is the fact that the sample is always tilted in a small amount, giving rise to change in focus, i.e. change in intensity at the monochromator slits, between the lowest and highest positions. This can be compensated by actively changing the focus during the scan.

4.4 PLE mapping

Once an optically active nanotube is found, the next step is to assign its chirality. For this I recorded a PLE map using the GUI described in Chap. 3. Fig. 4.8 c) shows a typical PLE map recorded on the NRC sample. The emission wavelength is recorded in the range 1200 nm - 1600 nm and the excitation is swept from 750 nm to 900 nm with 5 nm steps. The integration time is 2 seconds. Below 1270 nm, the emission from the Si substrate is clearly visible. The maximum

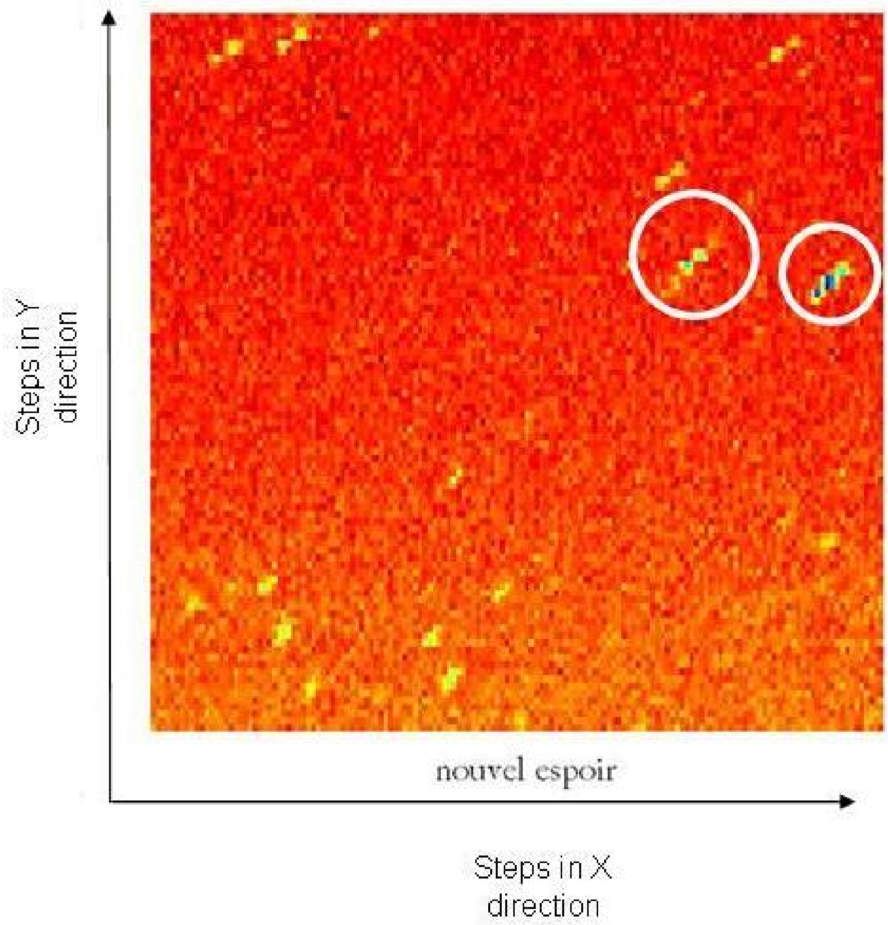

FIGURE 4.7: Mapped samples with possible nanotubes

intensity for the nanotube is recorded at an emission wavelength of about 1375 nm and for an excitation wavelength of about 855 nm.

From these two values, it is possible to assign the chirality of the nanotube. Indeed, Bachilo et al. studied SWCNTs suspended in micelles and could provide assignment tables from a comparison of their results with Raman spectroscopy on individual SWCNTs [30]. This table can be found in Appendix C. However, it is not possible yet to assign our nanotube. Indeed, it has been reported that the direct environment of individual nanotubes shifts the E_{22} and E_{11} transitions.

Lefebvre et al. found that emission peaks are blueshifted by 28 meV on average for the suspended nanotubes as compared to the encapsulated nanotubes. Similarly, the resonant absorption peaks at the second set of van Hove singularities are blueshifted on average by 16meV. Considering this small shifts, I found the best match for the tube in Fig. 4.8 to be (11,6). The spectrum

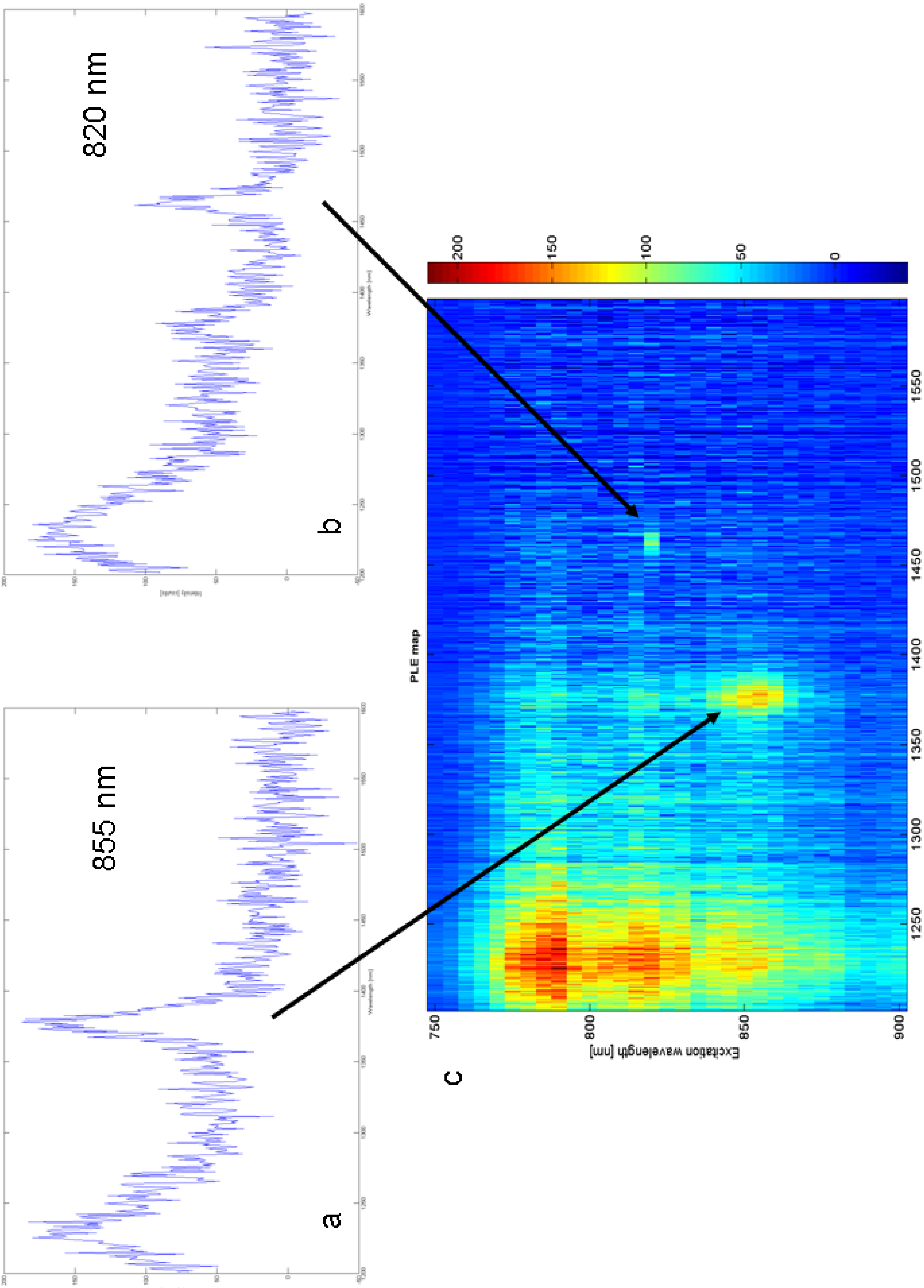

Figure 4.8: PLE map of a (11, 6) individual nanotube c), with cross-sections at 855 nm a) and 820 nm b)

of this tube recorded at an excitation wavelength of 855 nm is displayed in panel a). A small spot is also visible at an emission wavelength of 1510 nm and excitation wavelength of 820 nm. The spectrum displayed in panel b) at an excitation wavelength of 820 nm shows a peak shape similar to the one of a nanotube. The best match would then be a (12,5).

It is important to note that we recorded a higher resolution PLE map of the (11,6) nanotube prior to the shown map and the supposed (12,5) tube was not present. It might be a tube that became optically active for a reason we don't really understand. With a longer integration time and smaller steps in excitation wavelength, we would get a better quality PLE map. However, such long measurements would be strongly limited by the aging effect for tubes in ambient conditions we described earlier in section 4.2.

Chapter 5

Conclusion and Outlook

5.1 Conclusion and outlook

In this work, I measured the photoluminescence of individual single wall carbon nanotubes from different samples with a home made experimental setup. The most important part of my work was to implement routines in Matlab in order to automatize the search of optically active nanotubes as well as PLE mapping to assign the chirality of the tubes I found. I also modified the "hardware" part of the setup to enlarge the laser spot size on the sample in order to facilitate the tube searching process.

I observed an aging phenomenon characterized by a decay of the PL from individual tubes. The decay rate increased with the power density excitation. From the available literature, I attributed this effect to the relative humidity in the direct environment of the suspended nanotubes.

In the future, several improvements have to be made in order to allow more in-depth investigations of the optical properties of suspended nanotubes. This will be useful to design and study future nanotube-based optoelectronic devices, especially ultraclean nanotube devices developed in the Quantum Transport group using a new technology [31].

Some of the possible improvements are:

- Using closed loop piezo motors in order to have a reproducible positioning system. A beam scanner using galvo mirrors is to be considered as well. This would allow quicker record of spatial maps as well as a higher flexibility in the beam scanning.

- Include a system that allows a cleaner and dryer environment of the tubes, in order to minimize or even suppress the aging effect. This can be realized by the implementation of a vacuum chamber with gas inlets for flushing. This vacuum chamber could be easily converted to a cryostat allowing low temperature measurements.

- In this work, we did not perform polarization measurements. For this we need to setup polarizers and wave plates.

Appendix A

Program codes

A.1 RT_Setup.m

```
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
%                      Automization RT_setup @ Vlab
%
%                     Gilles Buchs and David Lakatos
%                       November 2009, TU Delft
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------


function varargout = RT_setup(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @RT_setup_OpeningFcn, ...
                   'gui_OutputFcn',  @RT_setup_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% ------------------------------------------------------------------------
%                    RT_setup_OpeningFcn
% ------------------------------------------------------------------------
%
% --- Executes just before RT_setup is made visible.
function RT_setup_OpeningFcn(hObject, eventdata, handles, varargin)

clc;
% Choose default command line output for RT_setup
handles.output = hObject;

% Default jog speed (3 = high speed)
handles.speed=3;

% Default Z speed (3 = high speed)
handles.speed_Z=3;

% Initialization Button Stop
handles.stop=0;

% Define action when one wants to close the GUI
set(handles.figure1,'CloseRequestFcn',@closeGUI);

% Define action when different radiobuttons in XY are selected
set(handles.Speed_Panel_buttongroup,'SelectionChangeFcn',...
    @Speed_Panel_buttongroup_SelectionChangeFcn);

% Define action when different radiobuttons in Z are selected
set(handles.Speed_Z_Panel_buttongroup,'SelectionChangeFcn',...
    @Speed_Z_Panel_buttongroup_SelectionChangeFcn);

% Define action when different togglebuttons in MODES are selected
set(handles.MODE_buttongroup,'SelectionChangeFcn',...
    @MODE_buttongroup_SelectionChangeFcn);

% Define action when different radiobuttons in Detector are selected
set(handles.Detector_buttongroup,'SelectionChangeFcn',...
    @Detector_buttongroup_SelectionChangeFcn);

% Initialization number of clicks on serial port gestion buttons
handles.Nb_click_open_XY=0;
handles.Nb_click_open_Z=0;
handles.Nb_click_close_XY=0;
```

```
handles.Nb_click_close_Z=0;

% Default wavelenght (index) value for map display
handles.sliderValue=1;

% Default value for detector pixels (InGas:512, Si:1340)
handles.pix=512;
handles.Max_slider=511;
handles.SliderStep=[0.00195 0.1];

% Update handles structure
guidata(hObject, handles);

% Close all open instruments
delete(instrfindall)
clear all
%
% ------------------------------------------------------------------------



% UIWAIT makes RT_setup wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% ------------------------------------------------------------------------
%                    RT_setup_OutputFcn
% ------------------------------------------------------------------------
%
% --- Outputs from this function are returned to the command line.
function varargout = RT_setup_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;
%
% ------------------------------------------------------------------------





% ------------------------------------------------------------------------
%                    Control serial ports (piezos & InGaAs)
% ------------------------------------------------------------------------
%
%
%
% --- Executes on button press in OpenXY.
```

```
function OpenXY_Callback(hObject, eventdata, handles)

handles.Nb_click_open_XY=handles.Nb_click_open_XY+1;
handles.Nb_click_close_XY=0;

if (handles.Nb_click_open_XY < 2)
    set(handles.XY_status,'String','OPEN');
    set(handles.XY_status,'BackGroundColor',[0.2,0.8,0]);
    set(handles.XY_status,'ForegroundColor',[1,1,1])
    guidata(hObject,handles);

    % Opens a serial communication object
    global S1
    S1=serial('COM5',...
              'BaudRate', 921600, ...
              'Parity', 'none', ...
              'DataBits',8, ...
              'StopBits', 1, ...
              'FlowControl','none',...
              'Terminator', 'CR/LF');
    fopen(S1)
else
    errordlg('Serial port already open','Error');
end
%
%
% --- Executes on button press in CloseXY.
function CloseXY_Callback(hObject, eventdata, handles)

handles.Nb_click_close_XY=handles.Nb_click_close_XY+1;
handles.Nb_click_open_XY=0;

if (handles.Nb_click_close_XY < 2)
    set(handles.XY_status,'String','CLOSED');
    set(handles.XY_status,'FontWeight','bold');
    set(handles.XY_status,'BackGroundColor',[1,0,0]);
    set(handles.XY_status,'ForegroundColor',[1,1,1])
    guidata(hObject,handles);
    %
    global S1
    fclose(S1);
    delete(S1);
    clear S1

else
    errordlg('Serial port already closed','Error');
end
%
```

```
%
% --- Executes on button press in OpenZ.
function OpenZ_Callback(hObject, eventdata, handles)

handles.Nb_click_open_Z=handles.Nb_click_open_Z+1;
handles.Nb_click_close_Z=0;

if (handles.Nb_click_open_Z < 2)
    set(handles.Z_status,'String','OPEN');
    set(handles.Z_status,'FontWeight','bold');
    set(handles.Z_status,'BackGroundColor',[0.2,0.8,0]);
    set(handles.Z_status,'ForegroundColor',[1,1,1]);
    guidata(hObject,handles);

% --- Opens a serial communication object
    global S2
    S2=serial('COM6',...
               'BaudRate', 921600, ...
               'Parity', 'none', ...
               'DataBits',8, ...
               'StopBits', 1, ...
               'FlowControl','none',...
               'Terminator', 'CR/LF');

    fopen(S2);

else
    errordlg('Serial port already open','Error');
end
%
%
% --- Executes on button press in CloseZ.
function CloseZ_Callback(hObject, eventdata, handles)

handles.Nb_click_close_Z=handles.Nb_click_close_Z+1;
handles.Nb_click_open_Z=0;

if (handles.Nb_click_close_Z < 2)
    set(handles.Z_status,'String','CLOSED');
    set(handles.Z_status,'FontWeight','bold');
    set(handles.Z_status,'BackGroundColor',[1,0,0]);
    set(handles.Z_status,'ForegroundColor',[1,1,1]);
    guidata(hObject,handles);

    global S2
    fclose(S2);
    delete(S2);
    clear S2
```

```
else
    errordlg('Serial port already closed','Error');
end


function Detector_buttongroup_SelectionChangeFcn(hObject,eventdata)

%retrieve GUI data, i.e. the handles structure
handles = guidata(hObject);

switch get(eventdata.NewValue,'Tag')   % Get Tag of selected object
    case 'InGaAs_radiobutton'
        handles.pix=512;
        handles.Max_slider=511;
        handles.SliderStep=[0.00195 0.1];
        display(handles.Max_slider)

    case 'Si_radiobutton'
        handles.pix=1340;
        handles.Max_slider=1339;
        handles.SliderStep=[0.001 0.1];
        display(handles.Max_slider)

    otherwise
       % Code for when there is no match.
       display 'Choose a speed'

end
%updates the handles structure
guidata(hObject, handles);

%
%
% --------------------------------------------------------------------------




% --------------------------------------------------------------------------
%                          MODE: Remote or Local
% --------------------------------------------------------------------------

function MODE_buttongroup_SelectionChangeFcn(hObject,eventdata)

global S1
global S2

switch get(eventdata.NewValue,'Tag')   % Get Tag of selected object
```

```
    case 'Remote_Mode_button'
        fprintf(S1,'MR')
        fprintf(S2,'MR')

    case 'Local_Mode_button'
        fprintf(S1,'ML')
        fprintf(S2,'ML')

    otherwise
       % Code for when there is no match.
       display 'Error'

end

% ------------------------------------------------------------------------






% ------------------------------------------------------------------------
%    Jog motion: manual acquisition of sample parameters (x_max_edit, y_max_edit)
% ------------------------------------------------------------------------
%
%
% --- Executes on button press in X_right.
function X_right_ButtonDownFcn(hObject, eventdata, handles)

if (handles.Nb_click_open_XY ≠ 0)
    global S1
    fprintf(S1,'MR')
    fprintf(S1,['1JA',num2str(handles.speed)])
    set(handles.X_right,'String','X +')
    set(handles.X_right,'BackgroundColor',[.2,0.8,0])

    set(gcf,'WindowButtonUpFcn',{@ButtonUpFcn_X_right,handles});
    guidata(hObject, handles);
else
    errordlg('Serial port need to be open','Error');
end

function ButtonUpFcn_X_right(src,eventdata,handles)

global S1
fprintf(S1,'1ST');

%read step number in X
fprintf(S1,'1TP');
```

```
temp_steps=fscanf(S1);
temp_length=length(temp_steps);
handles.Nb_steps_X=char(1:temp_length);
for i=1:(temp_length-3)
    handles.Nb_steps_X(i)=temp_steps(i+3);
end
set(handles.Pos_X_steps,'String',handles.Nb_steps_X)
set(handles.X_right,'String','X +')
set(handles.X_right,'BackgroundColor',[.925,0.914,0.847])


% --- Executes on button press in X_left.
function X_left_ButtonDownFcn(hObject, eventdata, handles)

if (handles.Nb_click_open_XY ~= 0)
    global S1
    fprintf(S1,'MR')
    fprintf(S1,['1JA-',num2str(handles.speed)])
    set(handles.X_left,'String','X -')
    set(handles.X_left,'BackgroundColor',[.2,0.8,0])

    set(gcf,'WindowButtonUpFcn',{@ButtonUpFcn_X_left,handles});
    guidata(hObject, handles);
else
    errordlg('Serial port need to be open','Error');
end

function ButtonUpFcn_X_left(src,eventdata,handles)
global S1
fprintf(S1,'1ST')

%read step number in X
fprintf(S1,'1TP');
temp_steps=fscanf(S1);
temp_length=length(temp_steps);
handles.Nb_steps_X=char(1:temp_length);
for i=1:(temp_length-3)
    handles.Nb_steps_X(i)=temp_steps(i+3);
end

set(handles.Pos_X_steps,'String',num2str(handles.Nb_steps_X))
set(handles.X_left,'String','X -')
set(handles.X_left,'BackgroundColor',[.925,0.914,0.847])


% --- Executes on button press in Y_up.
function Y_up_ButtonDownFcn(hObject, eventdata, handles)

```

```
if (handles.Nb_click_open_XY ≠ 0)
    global S1
    fprintf(S1,'MR')
    fprintf(S1,['2JA-',num2str(handles.speed)])
    set(handles.Y_up,'String','Y +')
    set(handles.Y_up,'BackgroundColor',[.2,0.8,0])

    set(gcf,'WindowButtonUpFcn',{@ButtonUpFcn_Y_up,handles});
    guidata(hObject, handles);
else
    errordlg('Serial port need to be open','Error');
end

function ButtonUpFcn_Y_up(src,eventdata,handles)
global S1
fprintf(S1,'2ST')

%read step number in Y
fprintf(S1,'2TP');
temp_steps=fscanf(S1);
temp_length=length(temp_steps);
handles.Nb_steps_Y=char(1:temp_length);
for i=1:(temp_length-3)
    handles.Nb_steps_Y(i)=temp_steps(i+3);
end

set(handles.Pos_Y_steps,'String',num2str(handles.Nb_steps_Y))
set(handles.Y_up,'String','Y +')
set(handles.Y_up,'BackgroundColor',[.925,0.914,0.847])


% --- Executes on button press in Y_down.
function Y_down_ButtonDownFcn(hObject, eventdata, handles)

if (handles.Nb_click_open_XY ≠ 0)
    global S1
    fprintf(S1,'MR')
    fprintf(S1,['2JA',num2str(handles.speed)])
    set(handles.Y_down,'String','Y -')
    set(handles.Y_down,'BackgroundColor',[.2,0.8,0])

    set(gcf,'WindowButtonUpFcn',{@ButtonUpFcn_Y_down,handles});
    guidata(hObject, handles);
else
    errordlg('Serial port need to be open','Error');
end

function ButtonUpFcn_Y_down(src,eventdata,handles)
```

```
global S1
fprintf(S1,'2ST')

%read step number in Y
fprintf(S1,'2TP');
temp_steps=fscanf(S1);
temp_length=length(temp_steps);
handles.Nb_steps_Y=char(1:temp_length);
for i=1:(temp_length-3)
    handles.Nb_steps_Y(i)=temp_steps(i+3);
end

set(handles.Pos_Y_steps,'String',num2str(handles.Nb_steps_Y))
set(handles.Y_down,'String','Y -')
set(handles.Y_down,'BackgroundColor',[.925,0.914,0.847])



function Speed_Panel_buttongroup_SelectionChangeFcn(hObject,eventdata)

%retrieve GUI data, i.e. the handles structure
handles = guidata(hObject);

switch get(eventdata.NewValue,'Tag')   % Get Tag of selected object
    case 'High_radiobutton'
        handles.speed=3;

    case 'Medium_radiobutton'
        handles.speed=2;

    case 'Low_radiobutton'
        handles.speed=1;

    otherwise
       % Code for when there is no match.
       display 'Choose a speed'

end
%updates the handles structure
guidata(hObject, handles);



% --- Executes on button press in Reset_X_Button.
function Reset_X_Button_Callback(hObject, eventdata, handles)

if (handles.Nb_click_open_XY ≠ 0)
    global S1
```

```
    fprintf(S1,'1ZP');
    set(handles.Pos_X_steps,'String','0');
    set(handles.Pos_X_um,'String','0');
    guidata(hObject, handles);
else
    errordlg('Serial port need to be open','Error');
end

% --- Executes on button press in Reset_Y_Button.
function Reset_Y_Button_Callback(hObject, eventdata, handles)

if (handles.Nb_click_open_XY ≠ 0)
    global S1
    fprintf(S1,'2ZP');
    set(handles.Pos_Y_steps,'String','0');
    set(handles.Pos_Y_um,'String','0');
    guidata(hObject, handles);
else
    errordlg('Serial port need to be open','Error');
end





% -------------------------------------------------------------------------
%     Focus Z motion
% -------------------------------------------------------------------------
%
%
% --- Executes on button press in Z_up.
function Z_up_ButtonDownFcn(hObject, eventdata, handles)

if (handles.Nb_click_open_Z ≠ 0)
    global S2
    fprintf(S2,'MR')
    fprintf(S2,['2JA',num2str(handles.speed_Z)])
    set(handles.Z_up,'String','Z +')
    set(handles.Z_up,'BackgroundColor',[.2,0.8,0])

    set(gcf,'WindowButtonUpFcn',{@ButtonUpFcn_Z_up,handles});
    guidata(hObject, handles);
else
    errordlg('Serial port need to be open','Error');
end

function ButtonUpFcn_Z_up(src,eventdata,handles)
global S2
```

```
fprintf(S2,'2ST')

%read step number in Z
fprintf(S2,'2TP');
temp_steps=fscanf(S2);
temp_length=length(temp_steps);
handles.Nb_steps_Z=char(1:temp_length);
for i=1:(temp_length-3)
    handles.Nb_steps_Z(i)=temp_steps(i+3);
end

set(handles.Pos_Z_steps,'String',num2str(handles.Nb_steps_Z))
set(handles.Z_up,'String','Z +')
set(handles.Z_up,'BackgroundColor',[.925,0.914,0.847])



% --- Executes on button press in Z_down.
function Z_down_ButtonDownFcn(hObject, eventdata, handles)

if (handles.Nb_click_open_Z ≠ 0)
    global S2
    fprintf(S2,'MR')
    fprintf(S2,'2SU-43');
    fprintf(S2,['2JA-',num2str(handles.speed_Z)])
    set(handles.Z_down,'String','Z -')
    set(handles.Z_down,'BackgroundColor',[.2,0.8,0])

    set(gcf,'WindowButtonUpFcn',{@ButtonUpFcn_Z_down,handles});
    guidata(hObject, handles);
else
    errordlg('Serial port need to be open','Error');
end

function ButtonUpFcn_Z_down(src,eventdata,handles)
global S2
fprintf(S2,'2ST')

%read step number in Z
fprintf(S2,'2TP');
temp_steps=fscanf(S2);
temp_length=length(temp_steps);
handles.Nb_steps_Z=char(1:temp_length);
for i=1:(temp_length-3)
    handles.Nb_steps_Z(i)=temp_steps(i+3);
end

set(handles.Pos_Z_steps,'String',num2str(handles.Nb_steps_Z))
```

```
set(handles.Z_down,'String','Z -')
set(handles.Z_down,'BackgroundColor',[.925,0.914,0.847])


% --- Select speed for Z motion.
function Speed_Z_Panel_buttongroup_SelectionChangeFcn(hObject,eventdata)

%retrieve GUI data, i.e. the handles structure
handles = guidata(hObject);

switch get(eventdata.NewValue,'Tag')   % Get Tag of selected object
    case 'Low_Z_radiobutton'
        handles.speed_Z=1;

    case 'Medium_Z_radiobutton'
        handles.speed_Z=2;

    case 'High_Z_radiobutton'
        handles.speed_Z=3;

    otherwise
       % Code for when there is no match.
       display 'Choose a speed'

end
%updates the handles structure
guidata(hObject, handles);


% --- Executes on button press in Reset_Z_Button.
function Reset_Z_Button_Callback(hObject, eventdata, handles)

if (handles.Nb_click_open_Z ≠ 0)
    global S2
    fprintf(S2,'2ZP');
    set(handles.Pos_Z_steps,'String','0');
    guidata(hObject, handles);
else
    errordlg('Serial port need to be open','Error');
end

% --------------------------------------------------------------------------
```

```
% -------------------------------------------------------------------------
%                           Sample mapping
% -------------------------------------------------------------------------
%
%
% --- Executes on button press in Stop_button.
function Stop_button_ButtonDownFcn(hObject, eventdata, handles)

if (handles.Nb_click_open_XY ≠ 0)
    global S1
    handles.stop=1;
    fprintf(S1,'RS');
    set(handles.Stop_button,'String','Stop')
    set(handles.Stop_button,'BackgroundColor',[1,0,0])
    set(handles.Run_button,'BackgroundColor',[0.925,0.914,0.847])

    set(gcf,'WindowButtonUpFcn',{@ButtonUpFcn_Stop,handles});
        guidata(hObject, handles);

end
function ButtonUpFcn_Stop(src,eventdata,handles)

set(handles.Stop_button,'String','Stop')
set(handles.Stop_button,'BackgroundColor',[.925,0.914,0.847])



% --- Executes on button press in Run_button.
function Run_button_ButtonDownFcn(hObject, eventdata, handles)

if (handles.Nb_click_open_XY ≠ 0)
    global S1
    set(handles.Run_button,'String','Run')
    set(handles.Run_button,'BackgroundColor',[0.2,0.8,0])

    % Creation of COM objects for communication with WinSpec
    objExp = actxserver ( 'WinX32.ExpSetup' );
    objDoc = actxserver ( 'WinX32.DocFile' );


    % Acquisition sample area
    X_max=str2num(get(handles.X_max_edit,'String'));
    Y_max=str2num(get(handles.Y_max_edit,'String'));

    % Acquisition mapping step size
    Inc_X=get(handles.Step_Size_X_edit,'String')
    Inc_Y=get(handles.Step_Size_Y_edit,'String')
```

```
    curDate = fix(clock);

    fid = fopen(strcat(get(handles.Edit_Path,'String'),'\_parameters.txt'), 'a');
    fprintf(fid, '%d-%d-%d %d:%d\r\n\r\nXmax> %d, Ymax>%d\r\n\r\nXstep> %s, Ystep> %s\r\n\r\n', ...
        curDate(1), curDate(2), curDate(3), curDate(4), curDate(5),...
        X_max, Y_max, Inc_X, Inc_Y);
    fclose(fid);

    X_pos=0;
    Y_pos=0;
    Y_index=0;
    stepCounter=0;

    handles.stop=0;

    set(handles.Pos_Map_X,'String',0);
    set(handles.Pos_Map_Y,'String',0);

    % Step size compensation
    Step_back=get(handles.Step_back,'String');
    fprintf(S1,['1SU-',Step_back]);


    maxDelay = 1000; % this corresponds to 10sec delay
    statusBit = 1;

    while (Y_pos ≤ Y_max)
        if mod(Y_index,2)==0
            while (X_pos < X_max)
                % incremant the step counter
                stepCounter=stepCounter+1;

                % record a spectrum at pos: (X_pos,Y_pos) now with extra
                % debugging

                if objExp.Start ( objDoc )
                    l = 0;

                    fprintf(1,'\nOpened file! Right now @ step %d \n', stepCounter);

                    while l < maxDelay
                        statusBit = objExp.GetParam ( 'EXP_RUNNING' );
                        if statusBit == 0
                             fprintf(1, ...
                                 '\nReady with creating spectrum, moving to next step');
                            break;
                        end
```

```
                            pause(0.01);
                            fprintf(1,'x');
                            l=l+1;
                        end

                        fprintf(1,'\n%d',l);
                        if l == maxDelay
                            fprintf(1, ['\n ----------------------------------------\n', ...
                                 'Maximum delay exceeded! In file %d'], ...
                                stepCounter);
                            objExp.Stop();
                            pause(0.1);
                        end

                        objDoc.Close;
                    end

                    % Move in +X direction by Inc_X
                    fprintf(S1,['1PR',Inc_X])
                    wait_ready1(S1)
                    X_pos=X_pos+str2num(Inc_X);
                    set(handles.Pos_Map_X,'String',num2str(X_pos))

                end
                Y_index=Y_index+1;
            else
                while (X_pos>0)
                    % incremant the step counter
                    stepCounter=stepCounter+1;

                    % record a spectrum at pos: (X_pos,Y_pos) now with extra
                    % debugging

                    if objExp.Start ( objDoc )
                        l = 0;
                        while l < maxDelay
                            statusBit = objExp.GetParam ( 'EXP_RUNNING' );
                            if statusBit == 0
                                break;
                            end
                            pause(0.01);
                            l=l+1;
                        end
                        if l == maxDelay
                            sprintf('Maximum delay exceeded! In file %d',num2str(stepCounter));
                        end

                        objDoc.Close;
```

```
                end

                % Move in -X direction by -Inc_X
                fprintf(S1,['1PR-',Inc_X])
                wait_ready1(S1)
                X_pos=X_pos-str2num(Inc_X);
                set(handles.Pos_Map_X,'String',num2str(X_pos))

            end
            Y_index=Y_index+1;
        end

                % record a spectrum at pos: (X_pos,Y_pos) now with extra
                % debugging

                if objExp.Start ( objDoc )
                    l = 0;
                    while l < maxDelay
                        statusBit = objExp.GetParam ( 'EXP_RUNNING' );
                        if statusBit == 0
                            break;
                        end
                        pause(0.01);
                        l=l+1;
                    end
                    if l == maxDelay
                        sprintf('Maximum delay exceeded! In file %d',num2str(stepCounter));
                    end

                    objDoc.Close;
                end

        % Move in +Y direction by Inc_Y
        fprintf(S1,['2PR',Inc_Y])
        wait_ready2(S1)
        Y_pos=Y_pos+str2num(Inc_Y);
        set(handles.Pos_Map_Y,'String',num2str(Y_pos))
    end
    set(handles.Run_button,'BackgroundColor',[0.925,0.914,0.847])

else
    errordlg('Serial port need to be open','Error');
end




% --- Wait until status S1 is ready
```

```
function wait_ready1(port)
fprintf(port,'1TS')
status=fscanf(port);
while (status(4) ≠ '0')
    fprintf(port,'1TS');
    status=fscanf(port);
end
%
% --- Wait until status S2 is ready
function wait_ready2(port)
fprintf(port,'2TS')
status=fscanf(port);
while (status(4) ≠ '0')
    fprintf(port,'2TS');
    status=fscanf(port);
end
%
% -------------------------------------------------------------------------





% -------------------------------------------------------------------------
%                      Display data
% -------------------------------------------------------------------------
%

function Edit_Path_Callback(hObject, eventdata, handles)








% --- Executes on slider movement.
function slider_E_Callback(hObject, eventdata, handles)

set(hObject,'Max',handles.Max_slider);
set(hObject,'SliderStep',handles.SliderStep);

%obtains the slider value from the slider component
handles.sliderValue = get(handles.slider_E,'Value');

%puts the slider value into the edit text components
slider_ind=fix(handles.sliderValue)+1;

```

```
set(handles.WV_Value,'String',num2str(slider_ind));

set(handles.WV,'String',num2str(handles.wl(slider_ind)));

set(handles.axes1,'NextPlot','replacechildren');

a=imagesc(handles.x,handles.y,handles.M(:,:,slider_ind));
set(a,'hittest','off');

colorbar;
xlabel('steps');
ylabel('steps');

% Update handles structure
guidata(hObject, handles);





% --- Executes on button press in Display_Button.
function Display_Button_Callback(hObject, eventdata, handles)

% Creation Matrix Data
X_max=str2num(get(handles.X_max_edit,'String'));
Y_max=str2num(get(handles.Y_max_edit,'String'));

Inc_X=str2num(get(handles.Step_Size_X_edit,'String'));
Inc_Y=str2num(get(handles.Step_Size_Y_edit,'String'));

col=X_max/Inc_X+1;
lines=Y_max/Inc_Y+1;

file_num=col*lines;

M=zeros(lines,col,handles.pix);

k=0;
for i=1:lines
    for j=1:col
        fid = fopen ( [strcat(get(handles.Edit_Path,'String'),'\spec_'),num2str(k),'.SPE']);
        %fid = fopen ( 'test_1.SPE' );
        tmp             = fread ( fid, 3263, 'int8');
        polynom_coeff   = fread ( fid,    6, 'double');
        tmp             = fread ( fid,  789, 'int8');
        data            = fread ( fid,       'int32' );
        fclose (fid);
        M(i,j,1:end)=data;
```

```
        k=k+1;
    end
end
% inverting each second line to respect the scan motion
B=M(2:2:end,:,:);
C=B(:,end:-1:1,:);
M(2:2:end,:,:)=C;

handles.M=M;

% Display map at wavelength index given by handles.sliderValue
handles.x = 0:Inc_X:X_max;
handles.y = 0:Inc_Y:Y_max;

axes(handles.axes1);

set(gca,'NextPlot','replacechildren');

a=imagesc(handles.x,handles.y,...
    handles.M(:,:,fix(handles.sliderValue)));

colorbar
xlabel('steps');
ylabel('steps');

handles.wl = polyval ( polynom_coeff(end:-1:1) , 1:length(data) )';

dataSpec = ones(1, 512);

dataSpec = squeeze(M(20,50,:));




figure(1)
plot(handles.wl, dataSpec);

% set(handles.axes1,'hittest','off');
set(a,'hittest','off');

guidata(hObject, handles);



% --- Executes on mouse press over axes background.
function axes1_ButtonDownFcn(hObject, eventdata, handles)

location = get(handles.axes1,'CurrentPoint');
```

```
x = round(location(1,1))
y = round(location(1,2))

X_max=str2num(get(handles.X_max_edit,'String'));
Y_max=str2num(get(handles.Y_max_edit,'String'));

Inc_X=str2num(get(handles.Step_Size_X_edit,'String'));
Inc_Y=str2num(get(handles.Step_Size_Y_edit,'String'));

fileNum = y * (X_max/Inc_X+1) + x

fid = fopen ( [strcat(get(handles.Edit_Path,'String'),'\spec_'),num2str(fileNum),'.SPE'] );
tmp             = fread ( fid, 3263, 'int8');
polynom_coeff   = fread ( fid,    6, 'double');
tmp             = fread ( fid,  789, 'int8');

if (handle.pix == 512)
    data            = fread ( fid,       'float' );
else
     data            = fread ( fid,       'int32' );
end


fclose (fid);

%TODO
% inverting each second line to respect the scan motion

% scale, poly coefficients...

scrsz = get(0,'ScreenSize');

figure('Position',[1 scrsz(4)/2 scrsz(3)/2 scrsz(4)/2])
plot(data);



% -------------------------------------------------------------------------


% -------------------------------------------------------------------------
%                   Close GUI function
% -------------------------------------------------------------------------
%
function closeGUI(src,evnt)

selection = questdlg('Are sure you want to close RT_setup?',...
                     'Close Request Function',...
```

```
                         'Yes','No','Yes');
switch selection,
   case 'Yes',
    delete(gcf)

   case 'No'
     return
end
% -------------------------------------------------------------------------
```

A.2 getbeam.m

```
% -------------------------------------------------------------------------
% -------------------------------------------------------------------------
%                        Subroutine to automatize beam
%                            are calculations @ Vlab
%
%                       David Lakatos, November 2009, TU Delft
% -------------------------------------------------------------------------
% -------------------------------------------------------------------------




function varargout = getBeam(varargin)
% GETBEAM M-file for getBeam.fig
%      GETBEAM, by itself, creates a new GETBEAM or raises the existing
%      singleton*.
%
%      H = GETBEAM returns the handle to a new GETBEAM or the handle to
%      the existing singleton*.
%
%      GETBEAM('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GETBEAM.M with the given input arguments.
%
%      GETBEAM('Property','Value',...) creates a new GETBEAM or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before getBeam_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to getBeam_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
```

```
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help getBeam

% Last Modified by GUIDE v2.5 14-Sep-2009 11:02:42

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @getBeam_OpeningFcn, ...
                   'gui_OutputFcn',  @getBeam_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before getBeam is made visible.
function getBeam_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

backgroundImage = importdata('headerklimt.jpg');
%select the axes
axes(handles.axes1);
%place image onto the axes
image(backgroundImage);
%remove the axis tick marks
axis off

backgroundImage = importdata('headerklimt2.jpg');
%select the axes
axes(handles.axes2);
%place image onto the axes
image(backgroundImage);
%remove the axis tick marks
axis off
```

```
% Choose default command line output for getBeam
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes getBeam wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = getBeam_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in viewButton.
function viewButton_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to viewButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

figure
fileName = strcat('C:\Documents and Settings\localadmin\Desktop\RT_setup',...
    get(handles.edit2,'String'));
a=imread(fileName);
b=a(:,:,1);
image(b);





% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1


% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fileName = strcat('C:\Documents and Settings\localadmin\Desktop\RT_setup',...
    get(handles.edit2,'String'));
a=imread(fileName);
b=a(:,:,1);

if get(handles.radiobutton1,'Value') == 1
    %the X coordinate should be used
    cut=b(:,str2num(get(handles.edit3,'String')));
elseif get(handles.radiobutton1,'Value') == 0
    cut=b(str2num(get(handles.edit4,'String')),:);
```

```
end

figure
plot(cut);



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'));
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

diff = abs(str2num(get(handles.edit5,'String')) - str2num(get(handles.edit6,'String')));

if get(handles.radiobutton1,'Value') == 1
   beamValue = 50/480* diff;

elseif get(handles.radiobutton1,'Value') == 0

end

%set(handles.text3,'String', num2str(beamValue));
set(handles.text3,'String', ['Beam size: ',num2str(beamValue),' um']);

optE = (beamValue/2)^2 * pi * 10^-2 * 4.98;

set(handles.text6,'String', ['Opt. energy: ',num2str(optE),' mW']);

handles.output = hObject;
guidata(hObject,handles);
```

A.3 plemap.m

```
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
%                      PLE map automatisation @ Vlab
%
%                     David Lakatos, November 2009, TU Delft
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------

function varargout = plemap(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @plemap_OpeningFcn, ...
```

```
                     'gui_OutputFcn',  @plemap_OutputFcn, ...
                     'gui_LayoutFcn',  [] , ...
                     'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before plemap is made visible.
function plemap_OpeningFcn(hObject, eventdata, handles, varargin)
global S1;

clc;

%deZIGNNN

backgroundImage = importdata('brecht.jpg');
%select the axes
axes(handles.axes2);
%place image onto the axes
image(backgroundImage);
%remove the axis tick marks
axis off

backgroundImage = importdata('brecht2.jpg');
%select the axes
axes(handles.axes3);
%place image onto the axes
image(backgroundImage);
%remove the axis tick marks
axis off

backgroundImage = importdata('brecht3.jpg');
%select the axes
axes(handles.axes4);
%place image onto the axes
image(backgroundImage);
%remove the axis tick marks
axis off

% Serial communication INIT
```

```
instrreset;

S1=serial('COM1');
set(S1, 'BaudRate', 115200, 'DataBits', 8, 'StopBits', 1, 'FlowControl', 'hardware');

fopen(S1);

%Motor initialisation

if not(libisloaded('USMCDLL'))
    loadlibrary('USMCDLL','USMCDLL.h')
end

global Devices_st;
global StartParam;
global State;
global Mode;

Devices_st = libstruct('USMC_Devices_st');
StartParam = libstruct('USMC_StartParameters_st');
State = libstruct('USMC_State_st');
Mode = libstruct('USMC_Mode_st');

%one member of the structure has to be initilised
Devices_st.NOD=0;

%Init command
retval=calllib('USMCDLL', 'USMC_Init', Devices_st);

%Get the status
Number_of_devices = Devices_st.NOD;

%every struct needs one field to be initialised
State.FullPower=1;
StartParam.Sdivisor=1;

retStart=calllib('USMCDLL', 'USMC_GetStartParameters', 0, StartParam);
retState=calllib('USMCDLL', 'USMC_GetState', 0, State);

if (retval==0)
    set(handles.motorStatus,'String','Initialised');
    set(handles.motorStatus,'BackGroundColor',[0.2,0.8,0]);
    set(handles.motorStatus,'ForegroundColor',[1,1,1])

    set(handles.temperature,'String',[num2str(State.Temp),'  C']);
    set(handles.motorPos,'String',num2str(State.CurPos));
end
```

```

handles.output = hObject;
guidata(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = plemap_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function from_Callback(hObject, eventdata, handles)
% hObject    handle to from (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of from as text
%        str2double(get(hObject,'String')) returns contents of from as a double


% --- Executes during object creation, after setting all properties.
function from_CreateFcn(hObject, eventdata, handles)
% hObject    handle to from (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function to_Callback(hObject, eventdata, handles)
% hObject    handle to to (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of to as text
%        str2double(get(hObject,'String')) returns contents of to as a double

```

```

% --- Executes during object creation, after setting all properties.
function to_CreateFcn(hObject, eventdata, handles)
% hObject    handle to to (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function intsec_Callback(hObject, eventdata, handles)
% hObject    handle to intsec (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of intsec as text
%        str2double(get(hObject,'String')) returns contents of intsec as a double


% --- Executes during object creation, after setting all properties.
function intsec_CreateFcn(hObject, eventdata, handles)
% hObject    handle to intsec (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global StartParam;
global State;
global S1;

%creating COM objects to communicate with ActiveX server
objExp = actxserver ( 'WinX32.ExpSetup' );
```

```
objDoc = actxserver ( 'WinX32.DocFile' );

%Set parameters in WInSpec
objExp.SetParam('EXP_FILEINCCOUNT',0);
objExp.SetParam('EXP_DATFILENAME','LastData');
% objExp.SetParam('EXP_DATFILENAME',['C:\Documents and Settings\localadmin\Desktop\'...
%      ,get(handles.path,'String')]);

from = str2num(get(handles.from,'String'));
to = str2num(get(handles.to,'String'));
step = str2num(get(handles.step,'String'));

Δ=abs(from-to);

cycles = (Δ/step)+1

pause(2);
StartParam.Sdivisor=1;

dataNorm = zeros(cycles,4);

for i=0:(cycles-1)

    %moving the amount needed
    %% for MIRA calib = 206.68
    %% for 3900 calib = -714.28
    calib = -714.28;

    calllib('USMCDLL','USMC_Start',0,(i*round(step*calib)),40.0,StartParam);



    % this corresponds to integration time * 1.5...if the
    % experimetn has not finished we move onto the next wavelength
    %maxDelay = 150);
    statusBit = 1;

    %wait to reach next wavelength
    pause(3);

    fprintf(S1,':POWER?');
    powTemp = str2num(fscanf(S1));

    %normalisation file create
    fid = fopen(['C:\Documents and Settings\localadmin\Desktop\',...
        get(handles.path,'String'),'\normalization.txt'], 'a');

    fprintf(fid, '%f %f\r\n', ...
```

```
        (from+i*step),powTemp);

    fclose(fid);


    dataNorm((i+1),1) = i;
    dataNorm((i+1),2) = powTemp;

    %reached wavelength, now let's record spectrum
    if objExp.Start ( objDoc )
        l = 0;
        while l < 100000
            statusBit = objExp.GetParam ( 'EXP_RUNNING' );
            if statusBit == 0
                break;
            end
            pause(0.01);
            l=l+1;
        end
%         if l == maxDelay
%             sprintf('Maximum delay exceeded! In file %d',num2str(cycles));
%         end

        objDoc.Close;
    end
    %updating information about the motor
    calllib('USMCDLL', 'USMC_GetState', 0, State);
    set(handles.temperature,'String',[num2str(State.Temp),'  C']);
    set(handles.motorPos,'String',num2str(State.CurPos));
    set(handles.curWL,'String',num2str(from+i*step));
end

%DISPLAY

M=zeros(cycles,512);
length(M);
a=1;

[z,zz] = textread(...
        ['C:\Documents and Settings\localadmin\Desktop\',...
        get(handles.path,'String'),'\normalization.txt'], ...
        '%f %f', cycles);

A = [z,zz];
sortrows(A,1);

excitation = A(:,1);
power = A(:,2);
```

```

maxPower = max(power);

axesExcitation = sort(excitation);

for k = 0:(cycles-1)
    fid = fopen (...
        ['C:\Documents and Settings\localadmin\Desktop\',...
        get(handles.path,'String'),...
    '\LastData',num2str(k),'.SPE']);

    tmp             = fread ( fid, 3263, 'int8');
    polynom_coeff   = fread ( fid,    6, 'double');
    tmp             = fread ( fid,  789, 'int8');
    intensity           = fread ( fid,       'float');

    fclose(fid);

    emission = polyval ( polynom_coeff(end:-1:1) , 1:length(intensity));

    for l = 1:512
         M(a,l) = (power(a) * intensity(l)) / maxPower;
    end
    a=a+1;
end

figure(1)
imagesc(emission,axesExcitation,M(:,:));
title('PLE map','fontsize',12,'fontweight','b')
xlabel('Emission wavelength [nm]')
ylabel('Excitation wavelength [nm]')

%END OF DISPLAY

handles.output = hObject;
guidata(hObject, handles);


function step_Callback(hObject, eventdata, handles)
% hObject    handle to step (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of step as text
%        str2double(get(hObject,'String')) returns contents of step as a double


% --- Executes during object creation, after setting all properties.
```

```
function step_CreateFcn(hObject, eventdata, handles)
% hObject    handle to step (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over motorStatus.
function motorStatus_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to motorStatus (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function setPos_Callback(hObject, eventdata, handles)
% hObject    handle to setPos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of setPos as text
%        str2double(get(hObject,'String')) returns contents of setPos as a double


% --- Executes during object creation, after setting all properties.
function setPos_CreateFcn(hObject, eventdata, handles)
% hObject    handle to setPos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
global StartParam
global State

calllib('USMCDLL','USMC_SetCurrentPosition',0,str2num...
    (get(handles.setPos,'String')));

calllib('USMCDLL', 'USMC_GetState', 0, State);

set(handles.temperature,'String',[num2str(State.Temp),'  C']);
set(handles.motorPos,'String',num2str(State.CurPos));


handles.output = hObject;
guidata(hObject, handles);



function moveEdit_Callback(hObject, eventdata, handles)
% hObject    handle to moveEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of moveEdit as text
%        str2double(get(hObject,'String')) returns contents of moveEdit as a double


% --- Executes during object creation, after setting all properties.
function moveEdit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to moveEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in moveSet.
function moveSet_Callback(hObject, eventdata, handles)
% hObject    handle to moveSet (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global State
global StartParam
```

```
%if the Wl has been set we can move
if(strcmp(get(handles.curWL,'String'),'?'))
else
    to=str2num(get(handles.moveEdit,'String'));
    from=str2num(get(handles.curWL,'String'));
    Δ=to-from;

    calllib('USMCDLL', 'USMC_GetState', 0, State);
    curPos=State.CurPos;

    %% for MIRA calib = 206.68
    %% for 3900 calib = -714.28
    calib = -714.28;

    steps2move = round(Δ * calib);
    StartParam.Sdivisor=1;

    calllib('USMCDLL','USMC_Start',0,(curPos+steps2move),40.0,StartParam);
    set(handles.curWL,'String',num2str(to));
end

handles.output = hObject;
guidata(hObject, handles);


function setWL_Callback(hObject, eventdata, handles)
% hObject    handle to setWL (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of setWL as text
%        str2double(get(hObject,'String')) returns contents of setWL as a double


% --- Executes during object creation, after setting all properties.
function setWL_CreateFcn(hObject, eventdata, handles)
% hObject    handle to setWL (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton3.
```

```
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

temp = str2num(get(handles.setWL,'String'));
set(handles.curWL,'String',num2str(temp));


function pathNorm_Callback(hObject, eventdata, handles)
% hObject    handle to pathNorm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pathNorm as text
%        str2double(get(hObject,'String')) returns contents of pathNorm as a double


% --- Executes during object creation, after setting all properties.
function pathNorm_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pathNorm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function path_Callback(hObject, eventdata, handles)
% hObject    handle to path (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of path as text
%        str2double(get(hObject,'String')) returns contents of path as a double


% --- Executes during object creation, after setting all properties.
function path_CreateFcn(hObject, eventdata, handles)
% hObject    handle to path (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
```

```
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Appendix B

Graphical User Interfaces

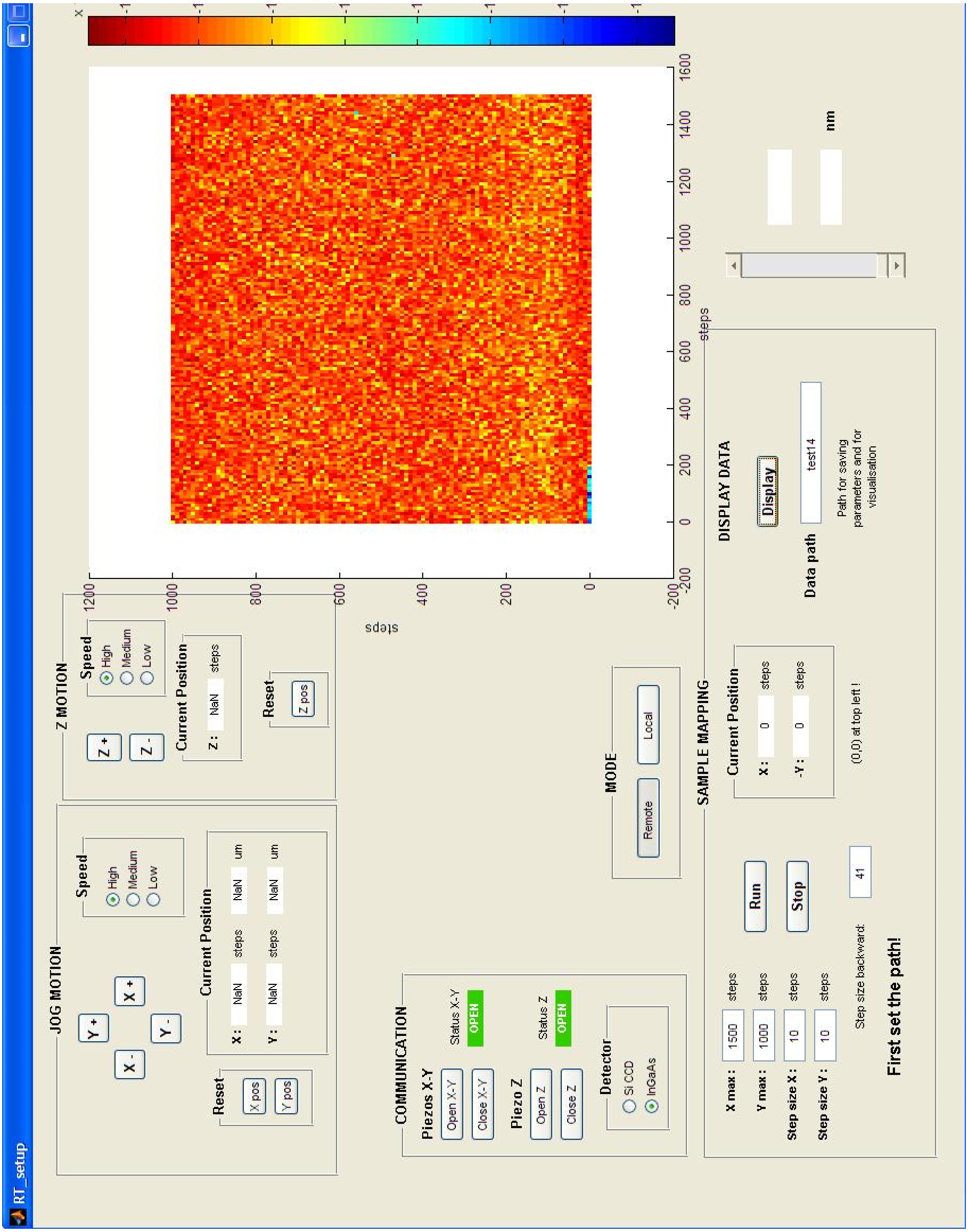

Figure B.1: GUI for the sample mapping automatization and stage control

FIGURE B.2: GUI for the beam area calculation

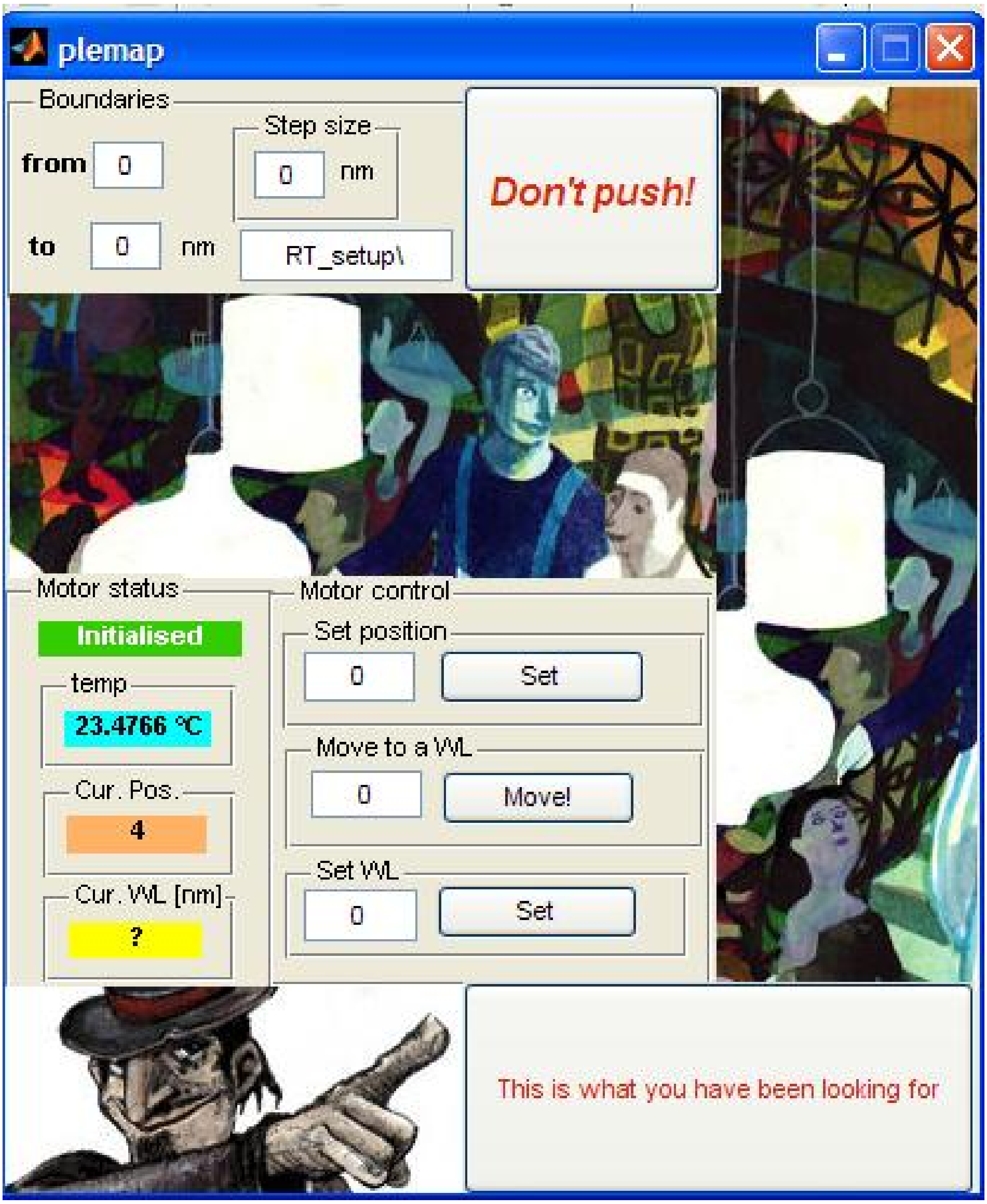

FIGURE B.3: GUI for the PLE mapping automatization

Appendix C

Table for assigning chiral indices

λ_{11} (nm)	λ_{22} (nm)	$h\nu_{11}$ (eV)	$h\nu_{22}$ (eV)	Assignment
833	483	1.488	2.567	(5,4)
873	581	1.420	2.134	(6,4)
912	693	1.359	1.789	(9,1)
952	663	1.302	1.870	(8,3)
975	567	1.272	2.187	(6,5)
1023	644	1.212	1.925	(7,5)
1053	734	1.177	1.689	(10,2)
1101	720	1.126	1.722	(9,4)
1113	587	1.114	2.112	(8,4)
1122	647	1.105	1.916	(7,6)
1139	551	1.088	2.250	(9,2)
1171	797	1.059	1.556	(12,1)
1172	716	1.058	1.732	(8,6)
1197	792	1.036	1.565	(11,3)
1244	671	0.997	1.848	(9,5)
1250	633	0.992	1.959	(10,3)
1250	786	0.992	1.577	(10,5)
1263	611	0.982	2.029	(11,1)
1267	728	0.979	1.703	(8,7)
1307	859	0.949	1.443	(13,2)
1323	790	0.937	1.569	(9,7)
1342	857	0.924	1.447	(12,4)
1372	714	0.904	1.736	(11,4)
1376	685	0.901	1.810	(12,2)
1380	756	0.898	1.640	(10,6)
1397	858	0.887	1.445	(11,6)
1414	809	0.877	1.533	(9,8)
1425	927	0.870	1.337	(15,1)
1474	868	0.841	1.428	(10,8)
1485	928	0.835	1.336	(13,5)
1496	795	0.829	1.559	(12,5)
1497	760	0.828	1.631	(13,3)
1555	892	0.797	1.390	(10,9)

FIGURE C.1: Table for assigning chiral indices to nanotubes based on their E_{11} and E_{22} energies

Bibliography

[1] Sumio Iijima. Synthesis of carbon nanotubes. *Nature*, 354:56–58, 1991.

[2] Marcus Freitag Phaedon Avouris and Vasili Perebeinos. Carbon-nanotube photonics and optoelectronics. *Nature Photonics 2, 341 - 350*, 2008.

[3] M Winger A Imamoglu A Hgele, C Galland. Photon antibunching in the photoluminescence spectra of a single carbon nanotube. *Physical review letters*, 2008.

[4] A Imamoglu C Galland. All-optical manipulation of electron spins in carbon-nanotube quantum dots. *Physical review letters*, 2008.

[5] X. Blase. J.-C. Charlier and S. Roche. Electronic and transport properties of nanotubes. *Rev. Mod. Phys. 79, 677*, 2007.

[6] S. K. Doorn X. Z. Liao Y. H. Zhao E. A.Akhadov M. A. Hoofbauer B. J. Roop Q. X. Jia R. C. Dye D. E. Peterson S. M. Huang J. Liu L. X. Zheng, M. J. OConell and Y. T. Zhu. Ultralong single-wall carbon nanotubes. *Nature Materials 3,673-676*, 2004.

[7] Rodney S. Ruoff and Donald C. Lorents. Mechanical and thermal properties of carbon nanotubes. *Elsevier Science*, 1995.

[8] E Lass B Wei PM Ajayan A Modi, N Koratkar. Miniaturized gas ionization sensors using carbon nanotubes. *Nature*, 2005.

[9] Q Wang M Lundstrom H Dai A Javey, J Guo. Ballistic carbon nanotube field-effect transistors. *Letters to Nature*, 1999.

[10] J. P. Lu. Elastic properties of carbon nanotubes and nanoropes. *Physical Rev. Lett.*, 1997.

[11] J. Maultzsch S. Reich and C. Thomsen. Tight-binding description of graphene. *Phys. Rev. B 66, 035412*, 2002.

[12] G. Dresselhaus M. S. Dresselhaus and A. Jorio. Unusual properties and structure of carbon nanotube. *Annu. Rev. Mater.*, 34:247, 2004.

[13] Chad B. Huffman Valerie C. Moore Michael S. Strano Erik H. Haroz Kristy L. Rialon Peter J. Boul William H. Noon Carter Kittrell Jianpeng Ma Robert H. Hauge R. Bruce Weisman Richard E. Smalley Michael J. O'Connell, Sergei M. Bachilo. Band gap fluorescence from individual single-walled carbon nanotubes. *Science, Vol. 297. no. 5581, pp. 593 - 596*, 2002.

[14] Strano M S et al. Reversible, band-gap-selective protonation of single-walled carbon nanotubes in solution. *J. Phys. Chem. B 107 697985*, 2003.

[15] Kiowski O Hennrich F Arnold K, Lebedkin S and Kappes. Matrix-imposed stress-induced shifts in the photoluminescence of single-walled carbon nanotubes at low temperatures. *Nano Lett. 4 234954*, 2004.

[16] Murakami Y Kishimoto S Maruyama S Ohno Y, Iwasaki S and Mizutani T. Excitonic transition energies in single-walled carbon nanotubes: dependence on environmental dielectric constant. *E Phys. Status Solidi b, 244 40025*, 2007.

[17] Homma Y Lefebvre J and Finnie. Bright band gap photoluminescence from unprocessed single-walled carbon nanotubes. *Phys. Rev. Lett. 90 217401*, 2003.

[18] Homma Y Lefebvre J, Fraser J M and Finnie. Photoluminescence from single-walled carbon nanotubes: a comparison between suspended and micelle-encapsulated nanotubes. *Appl. Phys. A 78 110710*, 2004.

[19] Louis E. Brus Tony F. Heinz Feng Wang, Gordana Dukovic. The optical resonances in carbon nanotubes arise from excitons. *Science, Vol. 308. no. 5723, pp. 838 - 841*, 2005.

[20] Yoshihiro Kobayashi Yoshikazu Homma and Toshio Ogino. Growth of suspended carbon nanotube networks on 100-nm-scale silicon pillars. *APPLIED PHYSICS LETTERS*, VOLUME 81, NUMBER 12:2261–62, 2002.

[21] Ogino T Homma Y, Kobayashi Y and Yamashita T. Growth of suspended carbon nanotube networks on 100-nm-scale silicon pillars. *Appl. Phys. Lett. 81 22613*, 2002.

[22] Shohei Chiashi Yoshikazu Homma and Yoshihiro Kobayashi. Suspended single-wall carbon nanotubes: synthesis and optical properties. *REPORTS ON PROGRESS IN PHYSICS*, 2009.

[23] Suzuki S and Kobayashi Y. Healing of low-energy irradiation-induced defects in single-walled carbon nanotubes at room temperature. *J. Phys. Chem. C 111 45248*, 2007.

[24] http://www.princetoninstruments.com/products/speccam/spec10.

[25] D. Mann et al. Electrically driven thermal light emission form individual single-walled carbon nanotubes. *Nature Nanotech 2, 33*, 2007.

[26] O. N. Torrens et al. Photoluminescence from intertube carrier migration in single-walled carbon nanotube bundles. *Nanolett. 6, 2864*, 2006.

[27] J. Lefebvre et al. *Phys. Rev. B 69, 075403*, 2004.

[28] M. W. B. Wilson. Investigations into the optical properties of individual, air-suspended, single-walled carbon nanotubes. *Master thesis, Queen's University, Kingston, Canada*, 2008.

[29] K. Kaminska et al. Real-time global raman imaging and optical manipulation of suspended carbon nanotubes. *Phys. Rev. B 73, 235410*, 2006.

[30] Carter Kittrell Robert H. Hauge Richard E. Smalley R. Bruce Weisman Sergei M. Bachilo, Michael S. Strano. Structure-assigned optical spectra of single-walled carbon nanotubes. *Science, Vol. 298. no. 5602, pp. 2361 - 2366*, 2005.

[31] G. A. Steele et al. Tunable few-electron double quantum dots and klein tunnelling in ultra-clean carbon nanotubes. *Nature Nanotechnology 4, 363 - 367*, 2009.

Acknowledgements

First of all I would like to thank my supervisors dr. Gilles Buchs and András Reichardt, without them this thesis would not have been possible.

Gilles, thank you for the countless hours spent in the lab and in front of the whiteboard squeezing physics into an engineer's mind and also for your patience while you were educating me.

András köszönöm mindazt a támogatást amit az évek során kaptam tŏled, mind a TDK dolgozatom, mind a szakdolgozat készïtése közben.

I would like to thank Leo Kouwenhoven and Val Zwiller for accepting me to the group and for providing guidance and helpful feedback on my work.

I also would like to acknowledge other people who helped me during my work. A not complete list of them: Martin, Martin, Reinier, Ilse ans Benny from Delft, Zsolt and Gábor from BME and Ildikó Varga form the Eramus Office in Budapest, who made it possible for me to spend 5 months in Delft.

I would also like to acknowledge nicotine, caffeine and alcohol as essential supporters of my thesis work.